João Pedro Pinto da Silva Ulisses

Mestrado em Multimédia da Universidade do Porto

Orientador: António Fernando Vasconcelos Cunha Castro Coelho (PhD)

Junho de 2014

Solução de geração procedimental de níveis de jogo

João Pedro Pinto da Silva Ulisses

Mestrado em Multimédia da Universidade do Porto

Aprovado em provas públicas pelo Júri:

Presidente: Rui Pedro Amaral Rodrigues (PhD)

Vogal Externo: Nelson Troca Zagalo (PhD)

Orientador: António Fernado Vasconcelos Cunha Castro Coelho (PhD)

Resumo

Nesta dissertação foi desenvolvida uma aplicação para a geração procedimental de níveis de jogo no âmbito de um jogo sério para a aprendizagem do ensino da programação. Esta aplicação baseia-se numa estrutura narrativa que foi desenvolvida no âmbito de uma outra dissertação de mestrado (que decorreu em simultâneo). Esta estrutura narrativa baseia-se num conceito base, o qual é desenvolvido como um enredo específico a partir de um plano de estudos definido pelo docente.

De acordo com a narrativa base, são geradas diversas salas e níveis que integram um conjunto de mecânicas específicas para dar forma ao plano de estudos definido, levando o estudante a resolver exercícios de programação para avançar no jogo.

Palavras-chave: Geração procedimental de conteúdos, Ensino da Programação, Jogos Sérios.

Abstract

An application was developed in this dissertation for the procedural generation of game levels for a serious game that has been developed for learning programming. This application is based on a narrative structure that was developed under another dissertation (held simultaneously). This narrative structure is based on a basic concept, which is developed as a specific storyline from a syllabus set by the teacher.

According to the narrative, which takes place in a spaceship, several rooms and levels are generated that include a set of specific mechanical to form a defined syllabus, leading the student to solve programming exercises to advance in the game.

Keywords: Procedural Content Generation, Learning Programming, Serious Games.

Agradecimentos

Agradeço ao meu professor António Coelho por me orientar, ao Ricardo Emanuel Ferreira Gonçalves, pela sua colaboração e ideias.

Agradeço também aos meus familiares, professores, colegas e amigos que direta ou indiretamente me motivaram a escolher este caminho e temáticas sobre os vídeo jogos.

João Pedro Pinto da Silva Ulisses

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

Tabela 1 - Abreviaturas e Símbolos

Json	JavaScript Object Notation
XML	eXtensible Markup Language
SVG	Scalable Vector Graphics
PCG	Procedural Content Generation
FEUP	Faculdade de Engenharia da Universidade do Porto
Fps	Frames per second (frames por segundo)
NPC	Non Playable Character
RPG	Roleplaying Game
MMORPG	Massive Multiplayer Online Roleplaying Game
UDK	Unreal Development Kit

Capítulo 1

Introdução

Nos dias de hoje existem cada vez mais jogos gerados procedimentalmente, podendo gerar todo o tipo de conteúdo em diferentes escalas, desde células básicas, até galáxias [6]. Existem muitas vantagens e desvantagens nos jogos gerados [6], mas existe muito pouca documentação sobre eles, e poucas formas de resolver os problemas, o que pode ser considerando como desvantagem.

Os jogos gerados procedimentalmente usando métodos PCG (Procedural Content Generation) permitem não só gerar quantidades enormes de dados, sejam mundos ou outro tipo de conteúdo com uma certa qualidade, essa qualidade é importante como será demonstrado, mas além disso o método PCG também permite fazer isto em pouco tempo [9], importante para o desenvolvimento de um jogo e também pode facilitar em vários tipos de teste do *software*.

Outras vantagens do uso de métodos PCG podem ir desde poupar espaço físico (memória) a custo de processamento. Por exemplo, Elite [4] (1984) oferece ao jogador mais de dois mil planetas para explorar numa disquete, o oposto também pode ser benéfico em que é conveniente poupar espaço na memória como terei oportunidade também de demonstrar e especificar em casos concretos qual a melhor solução usando uma métrica.

Os mundos gerados podem ser de grande dimensão, por exemplo o Daggerfall [24] é algumas vezes maior que Inglaterra quando posto à escala, também é maior que o mapa de Skyrim [24] que pertence à mesma série e é muito mais recente, isto deve-se também ao seu conteúdo muito mais detalhado, por exemplo os NPCs do Daggerfall têm um nome e dão algumas informações, no Skyrim apesar de em muito menor número de NPCs, têm personalidades e características específicas entre outras variáveis que tornam o conteúdo muito mais rico.

Jogos como o Minecraft [5] são muito conhecidos pelo termo “Sandbox” tendo um mundo gerado que pode ser infinito, um mundo aberto que pode ser destruído ou modificado.

Jogos de RPG e MMORPG muitas vezes além da dinâmica de mudar pequenas variáveis nos inimigos para criar inimigos novos e gerar assim novo conteúdo repetitivo para o jogador jogar é uma técnica muito conhecida, associado a isto normalmente estão as recompensas, normalmente items que são gerados com pequenas diferenças de atributos que dão ao personagem, como mais força e um valor, sendo normalmente estes *items*, quando muito raros e com valores elevados, muito valiosos na economia se o jogo for *online* e permitir trocas entre

jogadores, neste tipo de jogos é especialmente importante a geração de conteúdo, pois este é o que dá aos jogadores o que eles querem, que é jogar o jogo, mesmo que de certa forma seja repetitivo, se o conteúdo for suficientemente diferente ou incluir mecânicas novas associadas o jogo continua a ter sucesso, como é o caso de muitas séries de RPG e detestacar os MMORPGs em especial o World of Warcraft que tem mais de dez anos e com uma expansão nova a sair no final de 2014.

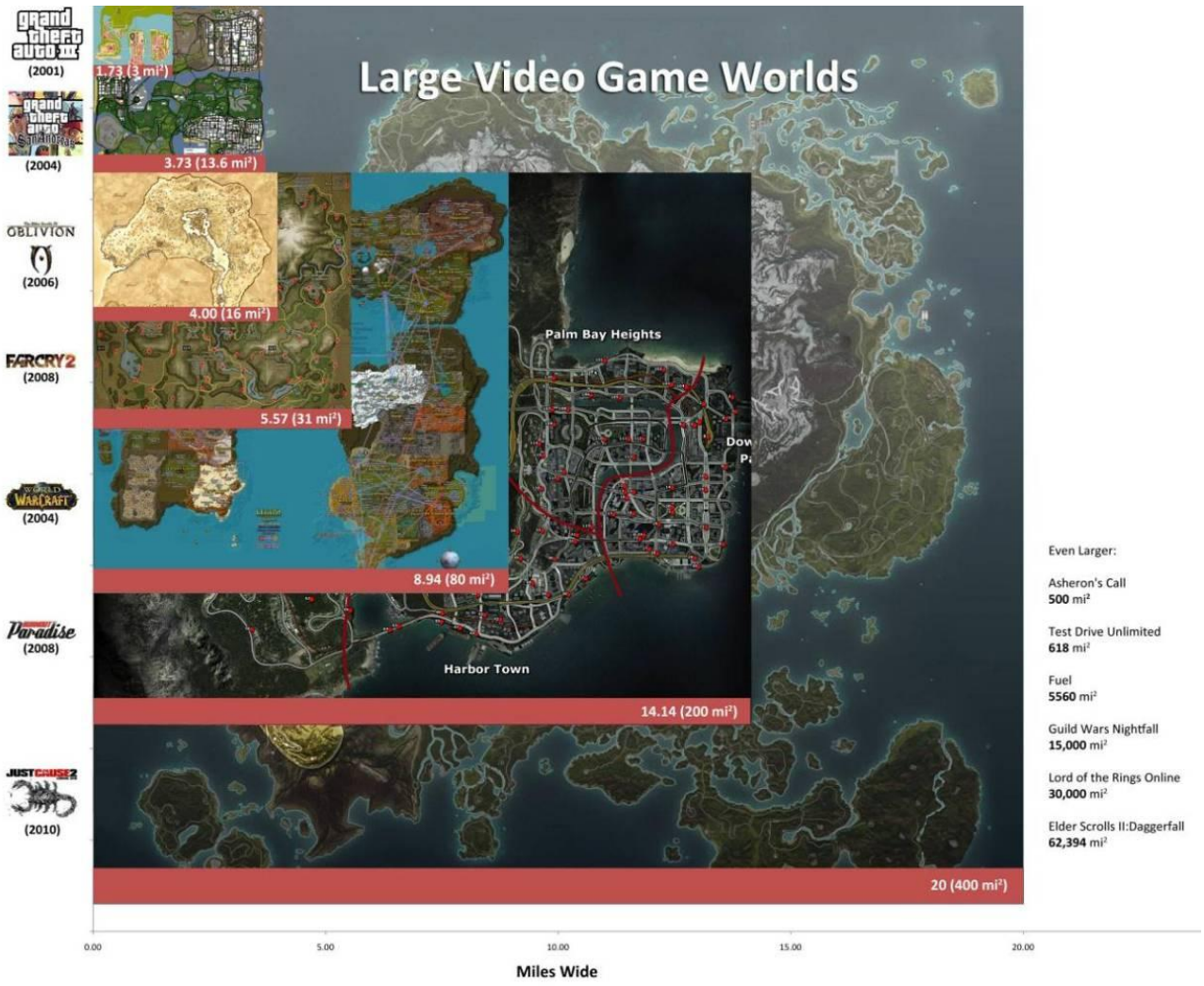


Figura 1 - Vários mapas de jogos gerados procedimentalmente.[23][24]

Existem vários tipos de conteúdos gerados e aproveitados, como é o caso da narrativa, e outros que eram para ser feitos e tratados, mas foram descartados, como é o caso do som e da música gerada procedimentalmente, porém aqui já existe muita coisa feita, por exemplo nos sons serem gerados e modificados com certos tons consoante varias variáveis, ou mudanças suaves ou agressivas de música conforme mudanças do jogo [9].

Alguns exemplos são:

- Rez (Sega) de 2001, um shooter ferroviário que liga firmemente a impressão visual (e aparência de inimigos) com a música composta e também feedbacks ações do usuário acusticamente.
- Electroplankton da Nintendo de 2005, onde o jogador interage de várias formas com o jogo para criar experiências de áudio e visual.
- O jogo Scratch-Off [7], que requer interação do usuário correspondente ao ritmo da música enquanto mistura para outra parte da música.
- O jogo Impossible Game é um jogo de plataforma gráfica simples, mas desafiador para consoles e telefones celulares que exige que o usuário a lidar com os mais variados obstáculos que são gerados de acordo com eventos musicais atuais.
- A série Bit.Trip por Gaijin Games apresenta o personagem principal comandante vídeo que tem de lidar com eventos do jogo principalmente com base no ritmo da música tocada em 6 jogos muito diferentes (atirador de ritmo, de plataformas).
- Wii Music gera automaticamente melodias com base em como o jogador move o controlador.
- BeatTheBeat [8] apresenta três minijogos (jogo baseado em ritmo torneira, atirador, e uma torre de defesa) que cada um usa a música como pano de fundo disponível e seus eventos recurso analisado como fonte para o jogo de criação do evento.

O mesmo acontece com as texturas, que podem ser geradas procedimentalmente, usando técnicas como Perlin Noise [13] e ainda o mesmo para os modelos e outros gráficos usando sistemas L [14] para plantas [15] [16] ou outros objetos, que podiam dar mais riqueza e qualidade do conteúdo gerado, e pode ser perfeitamente entregado no jogo Orion [1], mas a questão da narrativa é a que sempre teve problemas.

Existem métodos que conseguem estruturar o conteúdo de forma parecida de como foi feito no projeto Orion com a narrativa, usam elementos gráficos para criar bonecos, criar animações entre outras variáveis e conteúdo, exemplos disso são o tipo de ficheiro. json muitas vezes usado para animações e o SVG (Scalable Vector Graphics) para criar gráficos [29] sendo que o jogo pode escolher qual ficheiro SVG ler e decompô-lo criando e permitindo uma maior variedade e complexidade ao jogo.

Associado aos PCG estão os sensores biométricos que dão informação sobre o estado do jogador, podendo depois o jogo responder de certas formas obtendo um objetivo específico, o jogo também pode ser gerado em conformidade dessas variáveis ou dos objetivos pretendidos e ver se como o jogador se adapta.

1.1 Contexto/Enquadramento/Motivação

A ideia desta dissertação começou a partir do projeto Orion [1] um jogo sério [26] que necessitava de um mundo que se adaptasse ao jogo e à história, que fosse gerado a partir dos objetivos de aprendizagem, e para isso era preciso gerar um mapa a partir de um plano de exercícios e uma narrativa para criar um universo no jogo cujo objetivo seja o jogador aprender a programar.

Um dos objetivos era criar uma plataforma editável, para criar novas versões do jogo rapidamente e facilmente [3].

Outro projeto associado à dissertação é ainda algo que está em desenvolvimento sobre a documentação dos jogos, em que se pretende formular toda a documentação de forma a esta ser lida por um gerador que gera o jogo, permitindo poupar tempo no desenvolvimento e nos testes.

1.2 Projeto Orion

Esta dissertação parte de um projeto anterior, o projeto Orion, que consiste num jogo sério em Unity3D para o ensino da programação num nível universitário, podendo este ser alterado pelo docente.

"Os alunos aprendem muito mais quando estão a resolver problemas, planear algoritmos, e escrever programas do que assistir as aulas e tirando notas" [36], por isso, codificação e tarefas de programação devem ser a parte principal das mecânicas deste jogo [3].

Os principiantes em programação sentem-se mais atraídos por uma representação 3D do mundo do que uma versão 2D do mesmo. Estar em 3D não significa que o jogo vai ter uma apresentação extremamente realista, uma vez que vai exigir uma quantidade enorme de recursos e de tempo formar o pessoal [3].

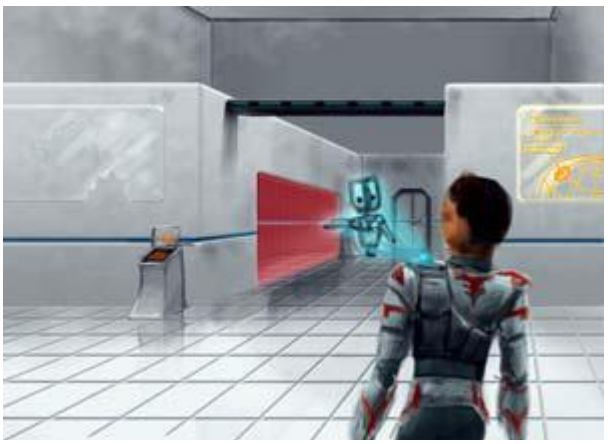


Figura 2 - Arte conceptual do mapa (desenhado por Enrique Kato). [3]

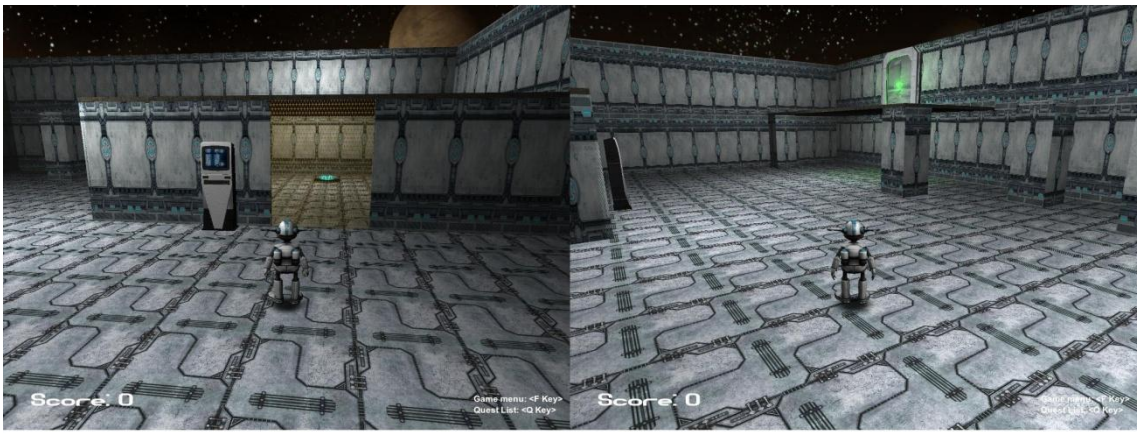


Figura 3 - Imagens do primeiro mapa do protótipo (por Ricardo Gonçalves).[2]

O jogador movimentava-se num mundo virtual 3D baseado numa matriz de células que podem estar vazias ou conter chão ou paredes. Esse mundo era também povoado por objetos que permitam ao jogador aceitar novos exercícios, conhecer a história e obter dicas sobre os exercícios. Para além desses objetos, deverão existir outros que potenciem a componente lúdica do jogo, como por exemplo campos de força ou teletransportadores. O mundo estava dividido em diversos mapas pequenos, tendo que existir portas que permitam a movimentação dos jogadores entre os vários mapas. De forma a poder jogar, o jogador terá que se autenticar e posteriormente selecionar um perfil, entre dez perfis, em que deseja jogar. Estes perfis são definitivos, isto é, não podem ser alterados. O objetivo destes perfis é dar a possibilidade de os jogadores poderem completar o jogo várias vezes, podendo até seguir diferentes vias de solução.

Relativamente aos exercícios, apenas se pode resolver um de cada vez e sempre que este é resolvido é atribuída uma pontuação ao jogador, tendo cada exercício uma pontuação específica.

No caso de errar a resolução, é incrementado o número de tentativas e quando finalmente este exercício for resolvido, a pontuação recebe uma penalização baseada no número de submissões erradas. O jogo poderá ser prolongado indefinidamente, a menos que seja adicionado um tipo de objeto especial cujo uso implique o fim do jogo [2].

Para criar os mapas a serem usados no protótipo teve que ser elaborado um editor de mapas simples, limitado a criar mapas de tamanho fixo de 20x2x20 [2].

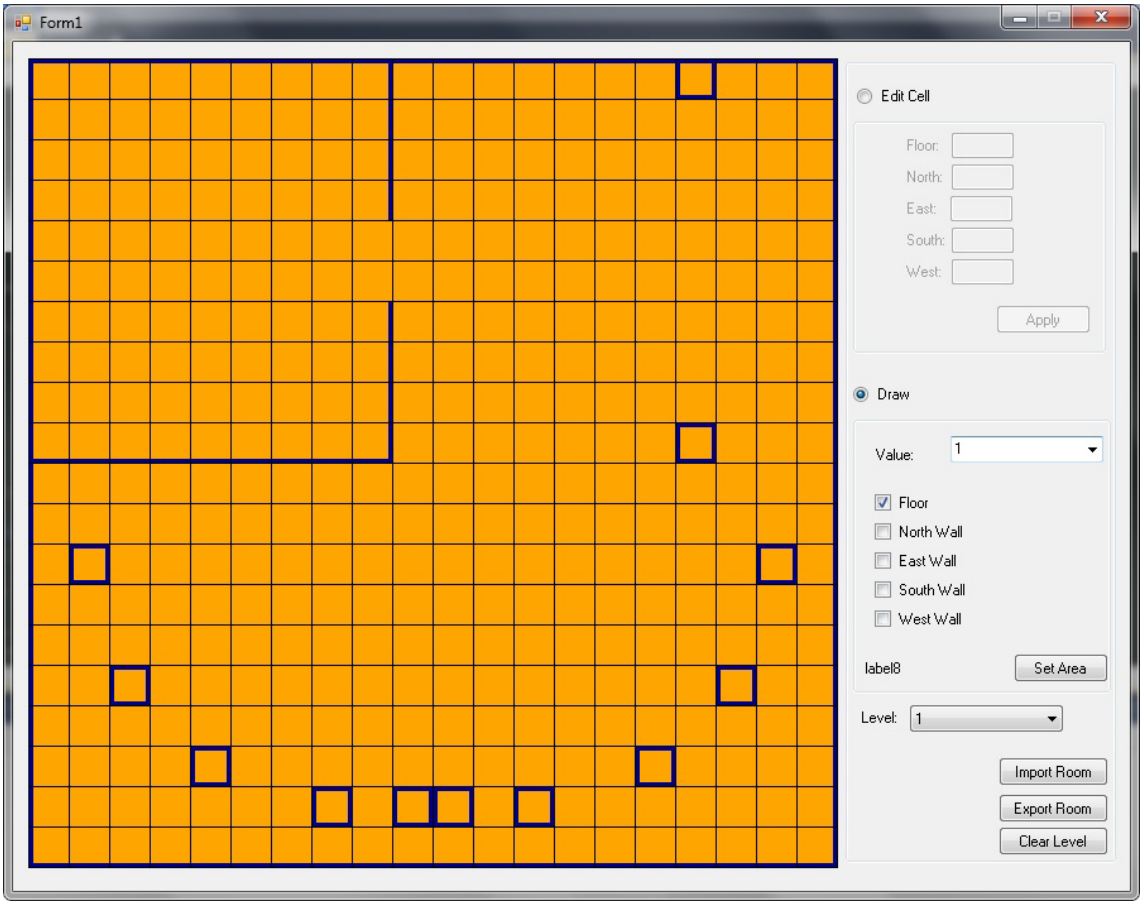


Figura 4 - Editor de mapas temporário (por Ricardo Gonçalves).[2]

anterior interage remotamente. Esta arquitetura segue a típica arquitetura cliente/servidor, no entanto, uma vez que parte fundamental da mecânica do jogo implica a resolução e compilação de exercícios de programação, surge a necessidade do uso de um sistema de correção assistida por computador, neste caso o DOMJudge [2].

Na imagem seguinte podem-se observar três componentes ligadas entre si. A componente de servidor, que controlará toda a plataforma. A componente cliente, a qual proporciona o jogo, desenvolvido com o motor Unity3D.

Finalmente o pacote do sistema DOMJudge, que através das suas capacidades de avaliação de código fonte permitirá à plataforma obter o resultado de soluções para as missões submetidas pelos jogadores [2].

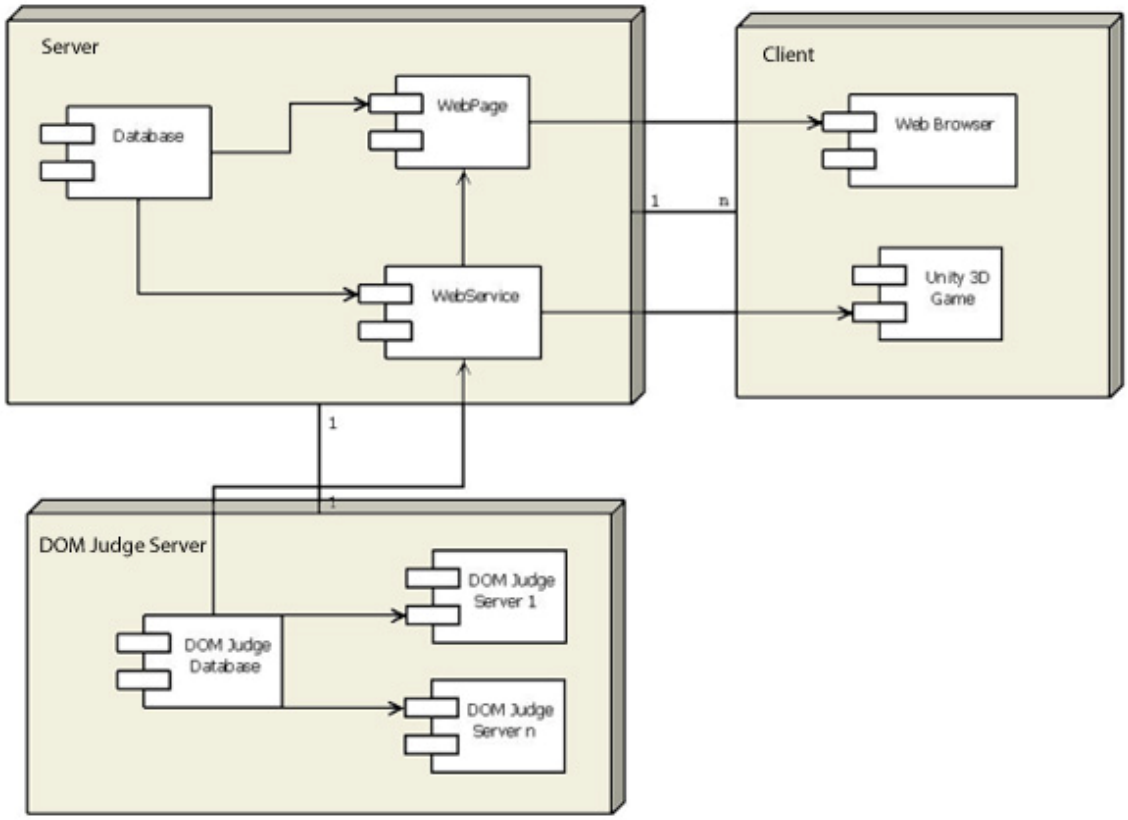


Figura 5 - Arquitetura geral. [2] [3]

1.3 Definição do Problema e Objetivos

O problema principal prende-se com a questão de como gerar um mundo de jogo, ligado a uma narrativa que se adapte a um plano de exercícios de programação evoluindo a abordagem em [25]. Mas com mais elementos da narrativa e um mundo mais complexo. Este jogo chamado Orion [1], possibilita que os exercícios de programação sejam inseridos como mecânicas de um jogo digital como forma de este processo ser divertido para os alunos que estejam a aprender a programar [3] [9].

Além disto o projeto deverá possibilitar a integração com projetos anteriores [1] e pensar nos futuros, sendo para isto necessário boa documentação, organização e compatibilidade entre várias linguagens.

No âmbito do problema enunciado, pretende-se uma solução genérica para jogos gerados procedimentalmente, e para tal definiram-se os seguintes objetivos:

- Selecionar um jogo para geração procedimental no âmbito para o ensino da programação;
- Identificar quais as melhores técnicas de geração procedimental;
- Melhorar a *performance* do jogo, identificando possíveis problemas e usar métricas que ajudem a melhorar o gameplay;
- Integrar o projeto com a narrativa, criando uma estrutura que o gerador consiga ler e criar um universo conforme essa narrativa;
- Criar algum dinamismo entre docente, narrativa, exercícios e jogador no jogo.
- Relacionar todas as variáveis, para uma maior complexidade harmoniosa no projeto, retirando e evitando variáveis absolutas que tornam o jogo estático.

1.4 Estrutura da dissertação

A dissertação está dividida em vários tópicos, começando por uma apresentação, do seu contexto e do projeto passando ao projeto desenvolvido e cada uma das suas componentes.

Será apresentado como podem ser gerados vários tipos de conteúdo, dando uma explicação e referindo a sua importância e qualidade do conteúdo [11] [21] para um jogo e respetivas métricas, mas também se vai mostrar mais detalhe em alguns tipos mais complexos, especificando partes, dando também solução para a narrativa que é muito problemática quando se usa PCG.

Será mostrada uma mais-valia na documentação de jogos que usem PCG, pois o seu conteúdo e a documentação estão relacionados, como será demonstrado.

A parte do jogo sério Orion que foi feito no âmbito desta dissertação, servirá de exemplo para a maior parte do conteúdo apresentado, mas apesar de ser um jogo especifico e cada caso ser um caso, o seu exemplo demonstrará que na maior parte dos jogos o mesmo se verifica, e nos raros casos em que seja diferente as métricas serão as mesmas podendo ter valores de importância diferentes, sendo os objetivos os mesmos, sendo que no fim serão mostrados resultados dos testes e conclusões.

Esse jogo faz parte de um projeto chamado Orion [1], um jogo sério [26] com o propósito de ensinar a programar de uma forma divertida [3].

Capítulo 2

Revisão Bibliográfica

Existem cada vez mais jogos gerados procedimentalmente, mas muito pouca documentação sobre eles, só em 2009 é que começou a surgir uma comunidade de investigação sobre este tema [9] com *workshops*, e só em 2011 saiu o primeiro documento que abordava este assunto. Até então só havia muita dedicação à inteligência artificial e computação gráfica, não havendo especial cuidado cientifico com a parte da geração, e muito menos, a formas de como resolver problemas neles existentes. Muitos problemas até aqui acontecem devido à enormidade dos dados multiplicando a sua complexidade e somando ao tipo de jogo, mas também existem outras razões para isso acontecer como por exemplo a má gestão dos recursos do jogo, para estes problemas vão ser mostradas soluções em diferentes casos.

Existem alguns casos de jogos em que se preocupavam com geração dos mapas e dungeons dos jogos mais antigos [17], estes jogos mais antigos eram baseados no Rogue de 1980 onde as cavernas eram geradas procedimentalmente, estes são normalmente chamados Roguelike, como um subtipo de jogo, em que normalmente se adiciona elementos de RPG, tornando-se então o Roguelike num subgénero dos jogos RPG.

Estes jogos no início tinham os gráficos representados por letras em ASCII e a sua dificuldade com conceitos como a morte permanente que eram evitados noutros jogos.

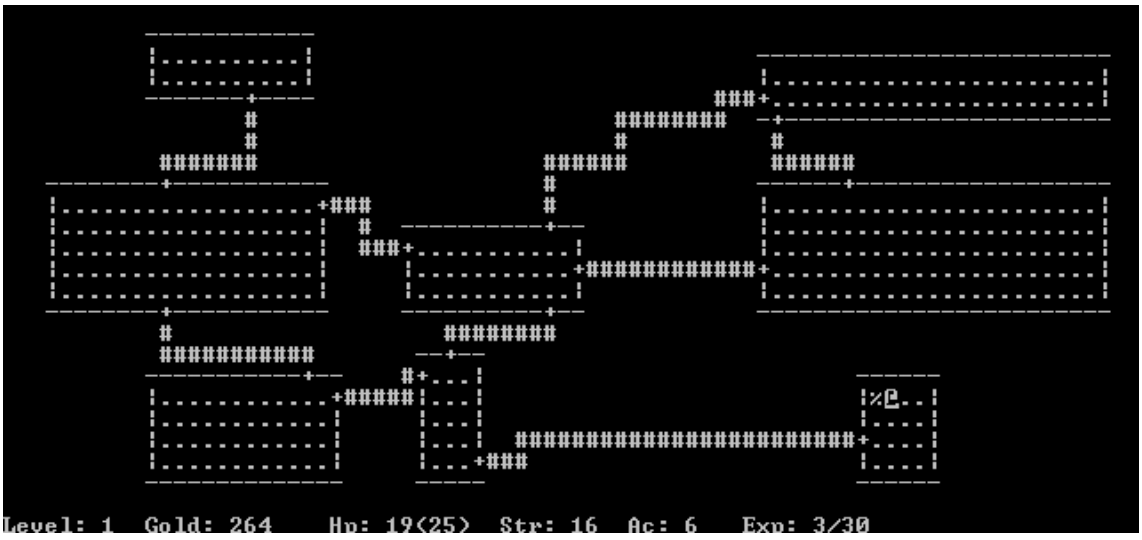


Figura 6 – Uma masmorra em Rogue

Foi graças a este tipo de jogos que mais tarde foram influenciadas as séries de jogos do Diablo que apesar do seu estilo ser um RPG de acção hack and slash, tem muitas semelhanças no que toca à geração de conteúdo e *gameplay* e em conceitos como a morte permanente em alguns modos.

Com a evolução dos computadores os jogos Roguelike voltaram a ficar famosos começando por volta de 2005 a ressurgir jogos no japão baseados em Animes, em RPG japoneses e "*spin-offs*" de séries para consolas portáteis, como foi o caso de Pokémon Mystery Dungeon da Nintendo entre outros.

Estes jogos baseavam-se em roguelike, e na sua grande maioria mantiveram o estilo apesar de começarem a retirar ou alterar alguns conceitos de jogadas por turno e morte permanente para agradar a um público mais jovem.

As características de um roguelike foram explicitamente definidas na Conferência Internacional de Desenvolvimento de Roguelike 2008, conhecido por "Berlin Interpretation" [37].

Alguns fatores relevantes para essa definição são:

- Jogos roguelike devem ter os níveis gerados procedimentalmente, embora possam incluir níveis estáticos também. O estilo normalmente incorpora quartos ligados por corredores, alguns dos quais podem pré-definir a existência de tesouros e monstros. Áreas abertas ou recursos naturais, como rios, também podem ser gerados.

- A identidade mágica dos itens varia entre as partidas. Objetos recém-descobertos oferecem apenas uma descrição física vaga, podendo ser aleatória e servir para quando há itens iguais aos já descobertos. Por exemplo, uma poção vermelha pode curar as feridas, ou então envenenar.

- O sistema do jogo é baseado em turnos em vez de estratégia em tempo real. O final do turno geralmente acontece no fim do movimento ou ação do jogador (efetuou um passo, usou um *item*). Após o fim do turno, o jogo processa o movimento dos monstros ou inimigos, efeitos progressivos (fome, dados de veneno, sangrar e recuperar vida).

- A maioria são jogos single-player. Em sistemas multi jogadores, os recordes são frequentemente compartilhados entre os jogadores. Alguns roguelikes concedem vestígios das antigas partidas nos jogos seguintes, exibindo fantasma ou lápide do personagem. Alguns jogos, como NetHack usa o personagem antigo como inimigo que aparece dentro da masmorra ou caverna.

- Roguelikes tradicionais implementam a morte permanente. Uma vez que o personagem morre, o jogador é obrigado a recomeçar o jogo. O recurso de "salvar o jogo" só deve ser usado para a suspensão da partida, não para recuperar jogos perdidos; o jogo salvo é excluído após a retomada da partida ou morte do personagem. Os jogadores podem contornar essa característica fazendo *backup* dos dados guardados (ato chamado de "save scumming") e até pode ir mais grave indo para o abuso de "save states" (abusando assim da sorte do jogo esperando sempre os melhores resultados possíveis), o que é considerado batota na comunidade destes jogos.

Um exemplo atual destas definições num jogo é o Dungeon Crawl Stone Soup, para PC e smartphones, que pode também ser jogado nos *browsers* podendo neste ultimo o resultado final ser partilhado entre outros jogadores (como é *online* é possível controlar os batoteiros), neste jogo pode ver-se todas as características de um *roguelike* clássico, o jogo é em código aberto e contínua em desenvolvimento [38] [31].

O projeto Orion é uma variante de roguelike, que tem as várias características idênticas como o facto de ser gerado procedimentalmente, os objetos tem nomes e descrições diferentes, o jogo foi feito para jogar sozinho (embora fosse fácil implementar o modo multijogador), porém como se trata de um jogo sério não convém existir a morte permanente nem o movimento por turnos, dando espaço para um estilo de jogo mais livre e de exploração, mais perdoável em caso de erros e mais agradável, sem que o jogador tenha que pensar tanto em cada ação que faz e não o sobrecarregando, guardando essas energias para quando estiver a resolver os problemas de programação.

Ou seja parte de entretenimento do jogo pode ser considerada um misto de roguelike com hack-slash do estilo do Diablo, com uma representação 3D dos mapas que podiam ser representados por tiles ou ASCII.

Cada vez mais os PCG são maiores e mais complexos, alguns problemas existentes nos PCG devem-se a falta da visão geral do que vai ser gerado, o que nos jogos causa por exemplo falhas visuais que podem ser observadas pelo jogador como objetos em sítios incorretos, falhas matemáticas onde se nota que uma escala está muito elevada ou baixa, ou ainda falhas que permitem aos jogadores explorar as mesmas, quer dando a perceber o algoritmo ou este estar vulnerável à manipulação para proveito do jogador.

Estes problemas podem vir do facto de normalmente requererem pessoas de áreas multidisciplinares, em que além de programarem têm de ter um conhecimento relacionado com o conteúdo a ser gerado, onde só o conhecimento de uma das áreas não chega para um bom resultado.

2.1 Conteúdo e qualidade

Os problemas mais comuns podem ser resolvidos recorrendo a boas práticas de programação, exceções e valores por definição entre outros que nesses casos mais inesperados o jogo deve ficar dentro dos limites apropriados, sendo que o custo das soluções destes problemas pode ser considerado baixo com o conhecimento certo, sendo muito benéfico face às outras soluções como o conteúdo mais feito à mão e conteúdo que é jogado uma ou duas vezes e depois deitado fora, pois torna este mais reutilizável.

Grande parte dos jogos evita conteúdo que só é usado uma vez, reutilizando-o, como em muitos jogos de RPG em que os inimigos aparecem com cores diferentes e estáticas diferentes, mas mudando poucos valores pode tornar o jogo repetitivo e chato.

O jogo gerado tem de ter em conta o divertimento do jogador, pois o divertimento é uma das partes mais importantes do jogo[20]. Tendo em conta as mecânicas de jogo e um bom *game design* [19] aliado, estes fazem com que o que venha a ser gerado seja o mais agradável possível. Para ser agradável, esse conteúdo tem de ter uma razão para existir sendo que essa pode estar relacionada com a qualidade do mesmo, ou seja se um jogador não encontrar razão para fazer um certo conteúdo, pode por em causa a consistência do jogo.

O papel da narrativa é muito relevante neste tipo de jogos, porém podem existir problemas de falta de sentido, falta de ligação com a narrativa com o jogo, os jogadores sentem falta de propósito nas suas tarefas, falta de contexto ou lógica e por vezes também acabam por ser repetitivas num curto espaço de tempo.

Um nível de um jogo Roguelike gerado pode ser avaliado segundo algumas métricas por exemplo controlo da área, exploração e balanço que ajudam a ver se os níveis gerados vão ser bons [35].

O conteúdo gerado nem sempre tem a qualidade desejada [11] [21], e algum desse conteúdo é opcional para estender o jogo [10], existem muitos fatores dependendo do tipo de conteúdo, mas para cada um existem métricas [21] que podem ajudar a obter a qualidade desejada.

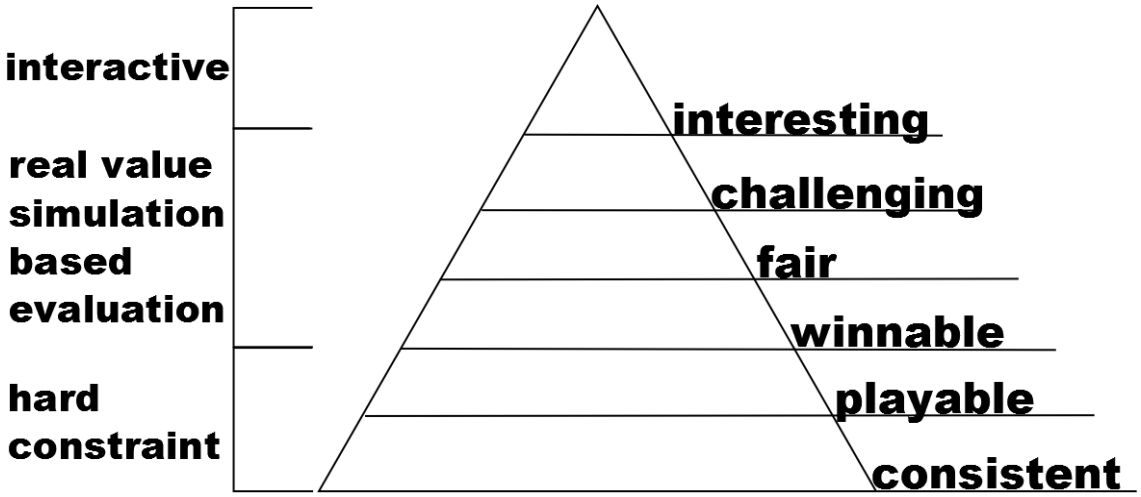


Figura 7 - Níveis de qualidade do conteúdo. [21]

A qualidade é uma das principais características procuradas nos jogos sérios.

2.2 Jogos sérios

Os jogos sérios conseguem associar a educação ao entretenimento, isto é, aprendizagem enquanto se faz uma atividade de um carácter mais lúdico, no entanto mantendo sempre presente que o objetivo é aprender. [2]

Projetos nesta área [2]:

- O Robocode é um jogo de programação, cujo objetivo lúdico é programar um tanque de guerra para enfrentar outros tanques desenvolvidos da mesma forma.
- O Wu's Castle é um jogo 2D desenvolvido no RPG Maker o qual permite aos seus utilizadores programar alterações em ciclos e matrizes de dados de forma interativa e visual.
- Alice é uma plataforma de programação 3D a qual permite criar animações para contar histórias e jogos.
- O M.U.P.P.E.T.S. é um sistema multiutilizador desenhado para permitir aos alunos desenvolver objetos em java, bem como criar robôs da mesma forma que o Robocode permite, no entanto com um grau de customização superior.
- A C-Sheep é uma mini linguagem de programação baseada na linguagem ANSI C, cujo objetivo é ajudar na aprendizagem dos princípios da ciência computacional e também da linguagem C.

Um factor importante que pode contribuir para o sucesso de um jogo sério, é os jogos terem a possibilidade de criarem ambientes virtuais onde podemos experimentar, tomar riscos e explorar várias vias para atingir objetivos e/ou resolver problemas [2].

Esses ambientes estão diretamente dependentes do conceito e objetivos que cada jogo tem, dando origem aos seguintes tipos de jogos sérios [2]:

- Simulação — Neste tipo de jogo sério o conceito principal é o de tentar reproduzir a realidade tão fielmente quanto possível de modo a proporcionar uma experiência realista que permita o jogador aprender pela prática. Um bom exemplo é o famoso Miscrosoft Flight Simulator, no qual é possível pilotar várias aeronaves reais por todo o planeta executando missões variadas, ou assumindo o papel de controlador de voo numa torre de controlo de um aeroporto;

- Estratégia e exercício de raciocínio — O xadrez talvez será o melhor exemplo de um jogo deste tipo, pois permite exercitar a nossa capacidade de antevisão e planeamento [2] podendo este ser considerado um jogo competitivo o que motiva muitos jogadores à semelhança de jogos de entretinimento;

- Treino físico — Com a saída da Wii no mercado das consolas, um novo tipo de jogo apareceu, a Wii Fit. Este jogo dispõe de vários sensores que permitem ao jogo seguir os movimentos do jogador e aconselhar correções em vários tipos de exercício físico, como Yoga, alguns desportos mais conhecidos e outras atividades;

- Estudo de áreas teóricas — Este tipo de jogo foca áreas científicas mais teóricas, como história, biologia, política, etc. Exemplo deste tipo de jogo é o PeaceMaker que coloca o jogador no papel de governante do estado Israelita ou Palestiniano e através de decisões governamentais tem que conseguir a atingir a paz.

- Estudo de áreas práticas — Este tipo de jogo foca áreas mais práticas, como a matemática, física, programação, etc. O Colobot é um jogo deste tipo, baseado num cenário fictício em que o planeta terra sofre um cataclismo terrível que força a humanidade a procurar um planeta capaz de sustentar vida. Para ajudar o jogador na sua missão, este tem robôs à sua disposição, os quais pode programar numa linguagem muito semelhante à linguagem de programação Java para executar várias tarefas, como recolha de recursos, construção ou defesa;

- Desenvolvimento de capacidades interpessoais — Os jogos deste tipo têm por função ajudar os jogadores a melhorar as suas capacidades de relacionamento com outros indivíduos e com as atividades que realiza em grupo. O melhor exemplo é o jogo NoviCraft o qual suporta vários jogadores e tem o objetivo de ajudar os seus jogadores a desenvolver as suas capacidades de cooperação e liderança através da resolução de puzzles e/ou problemas em grupo;

- Publicidade — Atualmente, este é possivelmente o tipo mais comum dentro dos jogos sérios e é caracterizado pelo objetivo de publicitar produtos e/ou serviços. Este tipo de jogo está presente principalmente em *banners* de páginas-web;

- Recrutamento — Estes jogos servem fundamentalmente para recrutar indivíduos, como é o caso do Americas's Army [2], um jogo de acção que coloca o jogador na posição soldado do exército dos Estados Unidos da América e dar uma imagem do que esperar na vida de soldado.

2.3 Tecnologias utilizadas no projeto

Para o projeto foi utilizado o Unity3D, escolhido devido às suas vantagens face a outros *game engines*. Na tabela abaixo é possível ver a comparação entre vários motores de jogo, mostrando que o Unity3D para este projeto é a melhor escolha, sendo a capacidade de poder compilar o código de várias linguagens a principal vantagem como será demonstrado.

Componente	**Unity3D**	**UDK**	**CryEngine 3**	**Torque 3D**
Suporte Gráfico	3D	3D	3D	3D
Animação	Sim	Sim	Sim	Sim
Áudio	2D & 3D	2D & 3D	2D & 3D	2D & 3D
Scripting	Javascript; C#.NET	UC	LUA	TS
Inteligência Artificial	Scripts	Scripts; Pathfinding; Decision Making	Scripts; Pathfinding; Decision Making	Scripts
Motor de física	Detecção de Colisões; Rigid Body; Vehicle Physics	Detecção de Colisões; Rigid Body; Vehicle Physics	Detecção de Colisões; Rigid Body; Vehicle Physics	Detecção de Colisões; Rigid Body; Vehicle Physics
Importação de conteúdo CAD	3ds max; maya	3ds max; maya	3ds max; maya	3ds max; maya
Editor de ambiente integrado	Sim	Sim	Sim	Sim
Código Fonte	Não	Não	Sim	Sim
Curva de aprendizagem	Média	Elevada	Média	Média
Documentação e tutoriais	Sim	Sim	Sim	Sim
Lincença	Gratuita	Gratuita	Gratuita[a]	$99
Suporte de Rede	Cliente/Servidor	Sim	Sim	Sim
Integração em web browser	Sim	Não	Não	Sim
Plataformas	PC; Mac; XBox; iOS	PC; iOS	PC; PSX; XBox; Gamecube	PC; Mac

a - Licença Educacional apenas para Universidades que façam uma aplicação que seja aceite pela Crytek. [2]

Tabela 2 - Comparação entre motores atuais. [2]

A linguagem de programação escolhida foi principalmente C#, visto que o projeto Orion também estava em C# podendo assim ser mais fácil integrar os projetos mais tarde. Contudo como o projeto está em Unity3D e existe uma terceira linguagem chamada Unityscript [34], que além de gerir todo o projeto a sua compilação, também permite a comunicação entre diferentes linguagens, como o Javascript, usado para instanciar objetos como o jogador e o robô e ainda a porta de um vídeo que estava feito em Javascript, a comunicação era necessária quando um objeto como por exemplo o robô era destruído o jogo tinha de o perceber através de um evento e comunicar com o código em C# para este saber que existia menos um robô, isto tudo permite com que o jogo seja gerado em C#, mas que seja perfeitamente possível gerar componentes e conteúdo em qualquer outra linguagem.

A documentação da narrativa foi feita em XML (eXtensible Markup Language) por permitir uma rápida, fácil e organizada leitura pelo gerador dos níveis como em [25], mas com mais elementos da narrativa e uma estrutura feita com elementos de *design* dessa área.

Existe a questão sobre memória versus processamento em que os jogos gerados procedimentalmente podem ter um grande papel, mas descartam muitas vezes a *performance* para o *hardware* quando estes podem e devem comunicar entre si para um maior rendimento.

Capítulo 3

Solução de geração procedimental de níveis de jogo

3.1 Introdução ao jogo Orion

O jogo Orion desenvolvido nesta dissertação é uma continuação independente do anterior no sentido prático, mas usa algumas ideias do projeto anterior como a mesma tecnologia e alguns recursos artísticos.

Utilizou-se também algum código desenvolvido a partir do tutorial "3D Platformer Tutorial" de alguns objetos como a camera, o jogador e o robô que foi editado e instanciado para uso no jogo Orion. Deste tutorial foram aproveitados alguns modelos gráficos como os do personagem do jogador, robô inimigo, nave e o teleporte, dado que se encaixavam no contexto da narrativa.

Qualquer um destes modelos ou texturas podem ser alterados sem qualquer problema para o jogo, no caso de um jogo comercial que fosse feito em equipas, estes podem ser alterados no documento do jogo onde podem ser substituídos pelos próprios artistas, poupando tempo ao resto da equipa principalmente aos programadores.

O jogo em si é similar ao que foi feito anteriormente, mas mais complexo, primeiro no mundo gerado, pois pode ser muito maior e depois pelas mecânicas de jogo que começam quando o jogador explora diferentes níveis, em que cada nível tem várias salas fechadas com portas, ligadas por corredores. Nas diversas salas o jogador encontrará vários objetos para interagir, como robôs inimigos, a sua nave, teleportes para outros níveis, terminais com exercícios, peças da nave, entre outros. Cada nível tem um ou vários objetivos podendo cada um destes ser diferente, podendo fazer com que o modo do jogador interagir com esses objetos seja diferente, como por exemplo fugir ou lutar.

3.2 A narrativa e a sua codificação XML

A narrativa do jogo foi desenvolvida em colaboração com o trabalho de mestrado de Ivo Brito [39], com o objetivo de incentivar o jogador a explorar os diversos níveis do jogo sério Orion, desenvolvido. Esta narrativa serve de fio condutor do jogo Orion, para o qual os desafios do jogo são gerados de acordo com o plano de exercícios.

Para haver uma integração da narrativa com a geração dos níveis de jogos, foi necessário desenvolver uma estrutura de dados em XML, sendo esta redefinida e melhorada em termos de *performance* enquanto se tratava da leitura do XML no gerador.

Foi criada a estrutura do ficheiro, guardando os tipos de informação necessária, permitindo informação adicional e foi-lhe colocado conteúdo, criando níveis, implementando mecânicas de jogo para cada como quem programa um nível e estes apontavam para objetos e exercícios dentro do nível. A título de exemplo, foi elaborado um plano de exercícios, para o qual, foram criados diálogos e cada um destes tinha informação de quando teria de aparecer no jogo, criando assim um ficheiro XML de exemplo de um jogo com uma narrativa, para ser usado como demonstração. Mais tarde estes ficheiros serão gerados com o *input* de um docente que dá um plano de exercícios e se quiser uma narrativa (ou está é gerada automaticamente também se não for apresentada uma pelo docente) que depois é lido pelo gerador do jogo.

Esta estrutura XML foi gradualmente atualizada e melhorada à medida que esta dissertação foi sendo desenvolvida, tendo vários casos possíveis. A título de exemplo, variáveis que podem existir ou não, ou estar vazias e todo esse tratamento de informação no gerador, estas variáveis podem aparecer ou não no ficheiro XML e o código do jogo tem de ser capaz de tratar disso e de gerar um resultado agradável. Ou seja, o jogo está preparado para ler informação incompleta e gerar a partir do que lê, porém quanto mais completa for, mais o jogo será gerado de acordo com quem preencheu o ficheiro XML.

Na narrativa existe um problema multidisciplinar em que é necessário existirem acordos para haver uma estrutura complexa o suficiente para o jogo gerado, que permita a esta ditar como este é gerado ou, o que se passa no mundo e criá-lo ou completá-lo, isto do ponto de vista de programação, o outro ponto de vista é para quem escreve a narrativa, que deve ser escrita de forma a agarrar o jogador e a motivá-lo a jogar. Juntar estes pontos de vista quando um dos lados nem sempre percebe o outro ou nem sempre são compatíveis, à primeira vista pode ser bastante complicado, porém se tudo for escrito ao pormenor permitindo várias hipótese sendo estas depois bem tratadas pelo gerado é possível. O trabalho de cada elemento da história deve estar bem identificado para depois ser utilizado nos sítios certos, que informação pode ter e que informação é que é opcional, com isto usou-se uma estrutura em XML, que é o resultado final para o gerador do jogo ler.

A leitura começa da seguinte forma, verifica se o ficheiro existe, se não existir, começou--se por gerar um mapa fixo, mas como isso podia confundir os resultados, o jogo agora não gera o mapa e deixa o jogador cair num buraco negro. Se o ficheiro existir ele começa a leitura.

Começando a leitura, verifica se o ficheiro começa com o elemento <game> e elemento a elemento vai ler o que tem dentro, sendo que a cada elemento vai ser comparado usando um switch que lê cada tipo de elemento XML possível, ao encontrar um e dependendo do seu nome e se tiver atributos dentro ele verifica se ele contém atributos e se os tiver passa os valores dos atributos para as respetivas variáveis, se tiver mais elementos dentro continua a leitura elemento a elemento, verificando sempre o que tem dentro até sair fora e chegar ao fim do game por consequência o fim do ficheiro.

Em termos de estrutura ficou dividido em diferentes formas:

Nome	Início	Fim	Conteúdo (elementos)
Game	<game>	</game>	leveldefinition, levels, objects, exercises, characters, dialogs

<game> (…) </game>

Elemento que indica o inicio e o fim, de tudo o que esta dentro do jogo.

Nome	Início	Fim	Conteúdo (atributos)
leveldefenition	<leveldefinition	</leveldefinition>	numrooms, distancelevels, currentlevel

<leveldefinition numrooms="30" distancelevels="500" currentlevel="0">
</leveldefinition>

Nome	Tipo	Exceção	Explicação
numrooms	Int	>= 0	Número de quartos
distancelevels	Int	> 0	Distância vertical entre os níveis
currentlevel	Int	> 0	Nível onde o jogador começa

leveldefinition é o elemento em que os seus atributos dão as definições do nível, sob forma de variáveis, por exemplo 30 quartos, em que a distancia entre os níveis será 500, e o jogador começa no nível 0, estas podem ser alteradas a qualquer momento, porém só devem ser alteradas pelo jogador se quiser um nível maior, ou por alguma razão específica na narrativa querer começar o jogo num certo nível, a distancia entre os níveis é uma questão apenas visual, podendo ter algum impacto na *performance*.

Nome	Início	Fim	Conteúdo (elementos)
levels	<levels>	</levels>	Level

<levels> (…) </levels>

Elemento onde são guardados os níveis, cada <level></level> esta dentro de levels.

Cada nível tem:

Nome	Início	Fim	Conteúdo (elementos)	Conteúdo (atributos)
Level	<level>	</level>	Objective	Idlevel, keyobjecttype, description, iddialog

<level idlevel="2" keyobjecttype="2" description="description of the level" iddialog="2">
(…)
</level>

Cada nível é representado desta forma, podendo a informação adicional não aparecer.

Atributos:

Nome	Tipo	Exceção	Explicação
idlevel	Int	> 0	Número do nível que se trata
keyobjecttype	Int, Nullable		Opcional, objeto mais importante para completar o nível
descrição	String		Opcional, descrição do nível
iddialog	Int, Nullable		Opcional, diálogo que aparece no início do nível

Cada nível contém um ou mais objetivos:

Nome	Início	Fim	Conteúdo (atributos)
objective	<objective	</objective>	type, idlevel_destination, idexercise, enemytype, numenemies

Atributos:

Nome	Tipo	Exceção	Explicação
Type	Int	> 0	Tipo de mecânica deste objetivo
Idlevel_destination	Int	> 0	Nível a qual o teleport vai enviar quando completa este objetivo
idexercise	Int, Nullable		Opcional, exercício para completar o nível
enemytype	Int, Nullable		Opcional, tipo de inimigos que aparecem no nível
Numenemies	Int, Nullable		Opcional, número de inimigos que aparecem no nível

<objective type="1" idlevel_destination="3" idexercise="2" enemytype="2" numenemies="1" />

Cada objetivo tem uma saída quando completo, sendo que um nível com vários objetivos pode ter vários caminhos opcionais.

<level idlevel="2" keyobjecttype="2" description="description of the level" iddialog="2">
<objective type="1" idlevel_destination="3" idexercise="2" enemytype="2" numenemies="1" />
<objective type="1" idlevel_destination="4" idobject="3" idexercise="3"/>
</level>

Aqui está um exemplo de um nível, neste caso o nível 2, que tem por objetivo apanhar um objeto do tipo 2, tem uma descrição e um diálogo quando entra no nível. Neste nível tem dois objetivos, completar um exercício por objetivo, no total de dois exercícios no nível, sendo que completando um ou outro desbloqueia um teleporte com um nível destino diferente, para desbloquear esse teleporte no primeiro caso tem de destruir um robô do tipo 2, ou para o caso do segundo teleporte apanhar um outro objeto, o id do objeto aponta para o objeto que está presente nesta estrutura, o mesmo acontece para os exercícios e os diálogos que podem ser requeridos para prosseguir no nível.

Podem existir tantos níveis quantos o docente ou quem cria o ficheiro XML desejar, ou também podem ser gerados aleatoriamente.

Podem também não ser indicadas certas variáveis, sendo estas nullable, podendo acontecer que são geradas, ou simplesmente não aparecem, como o caso dos robôs se não for dito o número de robôs ou for zero, eles não aparecem, o mesmo para os exercícios, objetos e níveis.

Nome	Início	Fim	Conteúdo (elementos)
objects	<objects>	</objects>	Object

<objects> (…) </objects> Contem os objetos do jogo para o personagem apanhar.

Cada objeto tem:

Nome	Início	Fim	Conteúdo (atributos)
object	<object	</object>	idtype, name

Atributos:

Nome	Tipo	Exceção	Explicação
idtype	Int	> 0	Id do objeto
name	String		Nome do objeto

<object idtype="3" name="Motor Quantum"></object Objeto com o nome Motor Quantum, este objeto pode aparecer num nível se for objetivo desse nível, ou opcional como foi demonstrado em cima em objective.

Nome	Início	Fim	Conteúdo (elementos)
exercises	<exercises>	</exercises>	Exercise

<exercises> (…) </exercises> Contem os exercícios de programação.

Cada exercício tem:

Nome	Início	Fim	Conteúdo (atributos)
exercise	<exercise	</exercise>	id, active, difficulty, description, iddialog

Atributos:

Nome	Tipo	Exceção	Explicação
Id	Int	> 0	Id do exercício
active	boolean		Opcional, se necessita de algo adicional para ativar o exercício ou não
difficulty	String		Opcional, diz a dificuldade do exercício, podendo este exercício aparecer mais facilmente se for fácil
description	String	!null	Descrição do exercício
iddialog	Int, Nullable		Opcional, diálogo depois de acabar o exercício, normalmente dá indicações sobre o que fazer

<exercise id="1" active="true" difficulty="easy"
description="int contaDigitos(string s) que percorre uma string e devolve o número de caracteres que são dígitos ('0' a '9')."
iddialog="2">
</exercise>

Exercício 1, fácil e pronto a ser feito, a descrição dizendo qual é o exercício que aparece no jogo para o jogador resolver e quando este é completado aparece o diálogo 2.

Nome	Início	Fim	Conteúdo (elementos)
characters	<characters>	</characters>	Character

<characters> (…) </characters>Contem os personagens do jogo.

Cada personagem tem:

Nome	Início	Fim	Conteúdo (atributos)
character	<character	</character>	id, name, type, standing

Atributos:

Nome	Tipo	Exceção	Explicação
Id	Int	> 0	Id do personagem
name	String	!null	Nome do personagem
Type	String		Opcional, diz o tipo ou raça do personagem
standing	String		Opcional, diz se o personagem é amigável, neutro ou inimigo

<character id="2" name="RA3" type="robot" standing="friendly">
</character>

Esta informação é útil por exemplo no diálogo para saber quem fala e para o jogador saber se é amigo ou inimigo.

Nome	Início	Fim	Conteúdo (elementos)
dialogs	<dialogs>	</dialogs>	Dialog

<dialogs> (…) </dialogs> contem vários diálogos

Cada diálogo tem:

Nome	Início	Fim	Conteúdo (elementos)	Conteúdo (atributos)
dialog	<dialog	</dialog>	entry	Id

Atributos:

Nome	Tipo	Exceção	Explicação
Id	Int	> 0	Id do diálogo

<dialog id="1"> (…) </dialog>

Contem vários entries onde é guardada cada fala ou momento do diálogo, cada diálogo tem um id para poder ser evocado.

Cada diálogo contém um ou mais entries:

Nome	Início	Fim	Conteúdo (atributos)
Entry	<entry	</entry>	id, text

Atributos:

Nome	Tipo	Exceção	Explicação
idspeaker	Int	> 0	Saber quem é a personagem que fala
Text	String	!null	Texto da fala

<entry idspeaker="1"
text="RA3? Relatório de danos.RA?">
</entry>

Numa das falas do jogo quando este começa, o jogador tem de pressionar T para ler a próxima fala que é o próximo elemento entry dentro do mesmo dialog, o jogador é aconselhado a usar a tecla T dentro do jogo.

O jogo só lê campos específicos identificados, se ele não encontrar campos que podiam existir, como robôs, então ele percebe que essa foi a escolha de quem escreveu o XML e não cria robôs, se ler coisas do tipo <naoler naoler="naoler"></naoler> mesmo que estejam num formado XML válido, o gerador não lê essa parte pois no gerador não foi preparado para tal, só vai ler certas variáveis já identificadas, mas gera o jogo com a informação bem identificada que obteve na mesma, isto caso quem escreve se possa ter enganado em algo, essa parte pode ser ignorada, contudo o jogo não é gerado de acordo com o XML se este ficheiro XML não estiver válido pelas normas do w3c[27].

Este conteúdo é recebido por listas, por exemplo no caso do nível:

```
private static List<InfoLevel> LevelInfoList = new List<InfoLevel>();
public static List<InfoLevel> out_LevelInfoList  { get { return LevelInfoList; } }
```

```
public class InfoLevel
{
	public int numLevelEnemies = 0; //total de inimigos neste nível
	public int numLevelExercise = 0;
	public int numLevelPiece = 0;
	public bool completeRobots = true;
	public bool completeExercise = false;
	public bool completePiece = false;
	public int idlevel; //"1"
	public int? keyobjecttype = null; //"1"
	public string description; //"description of the level"
	public int? iddialog = null; //"1"
	int objectivetype; //"1"


	public List<InfoObjective> objective = new List<InfoObjective>();

	public List<InfoObjective> out_objective { get { return objective; } }

}

public class InfoObjective
{
	public int type;
	public int idlevel_destination; //"2"
	public int? idobject = null; //"3"
	public int?	idexercise = null; //"1"
	public int? enemytype = null; //"0"
	public int? numenemies = null; //"1"
}
```

Existindo assim uma lista para guardar todos os níveis, e cada um destes elementos (um nível) tem uma lista com um ou vários objetivos As variáveis nullable não precisam de ser preenchidas isto para o caso de não serem mencionadas no documento XML.

A estrutura da narrativa pode ser modificada facilmente, os elementos nullable permitem flexibilidade.

O docente ou quem escreve a historia, irá ter um interface amigável que identificará o que está a escrever e colocará no local certo no ficheiro XML, preenchendo-o, e assim obtendo uma narrativa estruturada pronta a alimentar um jogo que pode ser gerado com base nessa narrativa.

As ambiguidades devem ser tratadas pela parte que gera o ficheiro XML e as contradições também, sendo que alguns tipos de contradição já são tratados no código do gerador do jogo, mas a ambiguidade como é um problema de especificação e o código do gerador do jogo está preparado para solucionar o problema ele pode faze-lo de forma não tão agradável quanto esperado, daí ser importante especificar o melhor possível para evitar ambiguidades, a menos que a ambiguidade seja pretendida para criar uma maior variação de possibilidades do jogo.

A documentação usada pode ser usada para outros componentes, tendo em conta as diferenças, mas de modo geral podendo proceder do mesmo modo, podendo assim a partir de um documento de *design* bem construído, ser possível junto com um gerador gerar o jogo a partir de um documento.

3.3 Geração procedimental de conteúdos

Vários conteúdos podem ser gerados, para um jogo existe muito conteúdo que é opcional, ou conteúdo que embeleza e enriquece o jogo. O conteúdo obrigatório é aquele que sem ele não poderia existir o jogo, como o mapa do jogo, o mundo e as mecânicas de jogo. Gerar estes mantendo uma relação entre o mundo e as suas mecânicas de jogo é complexo (variáveis relativas), mas é importante esses estarem relacionados entre si e não serem completamente independentes, para o resultado ser harmonioso, devem apenas ser independentes na sua componente funcional ou seja um objeto que seja instanciado depende do mundo e da sua posição, mas deve ser independente o suficiente para conseguir funcionar no sitio em que está.

O primeiro passo será criar um mundo virtual onde o jogo possa ser jogado, que representa as salas, os níveis e o seu conjunto tornando-se no ambiente 3D visível ao jogador.

3.3.1 Geração do mapa e níveis de jogo

O conteúdo gerado foi em maioria o mundo do jogo que estaria dependente de uma narrativa, criando um mapa de níveis cada um contendo quartos que o jogador pode explorar.

O jogo pode ser gerado de várias maneiras, dependendo do objetivo, sendo que no caso do Orion o jogo é 3D, mas é possível aplicar as mesmas regras para o 2D [12].

O algoritmo usado para a geração foi feito no âmbito do projeto Orion, mas pode ser alterado ou substituído como uma componente independente, sendo que apenas é importante informar ao jogo que gerador é que vai correr para gerar o mapa, para depois o desenhar e povoar. Existem muitos outros geradores de dungeons disponíveis na internet, alguns até *open source* como em [31] [32] [33], se num documento do jogo estiver estipulado qual o gerador a usar, este pode ser trocado a qualquer altura até numa fase final de desenvolvimento.

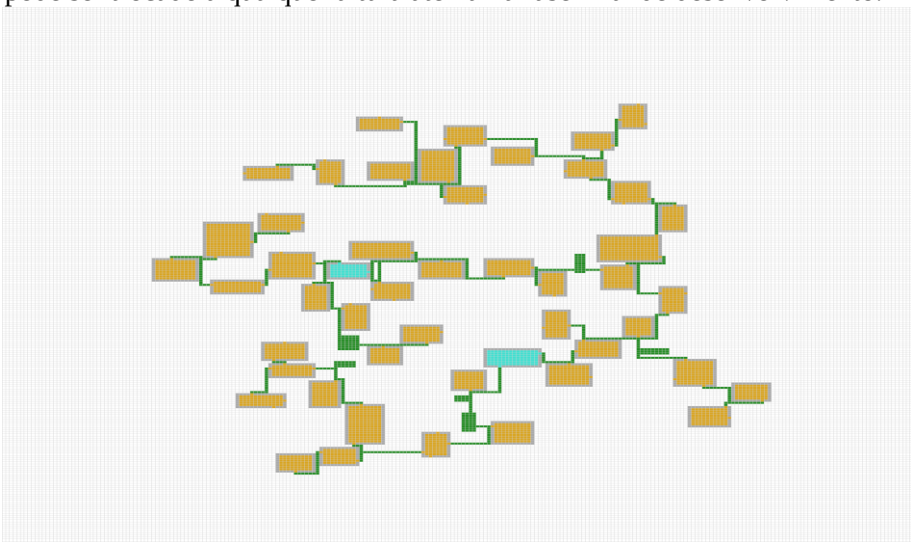


Figura 8 – Um nível gerado usando o gerador de mapas

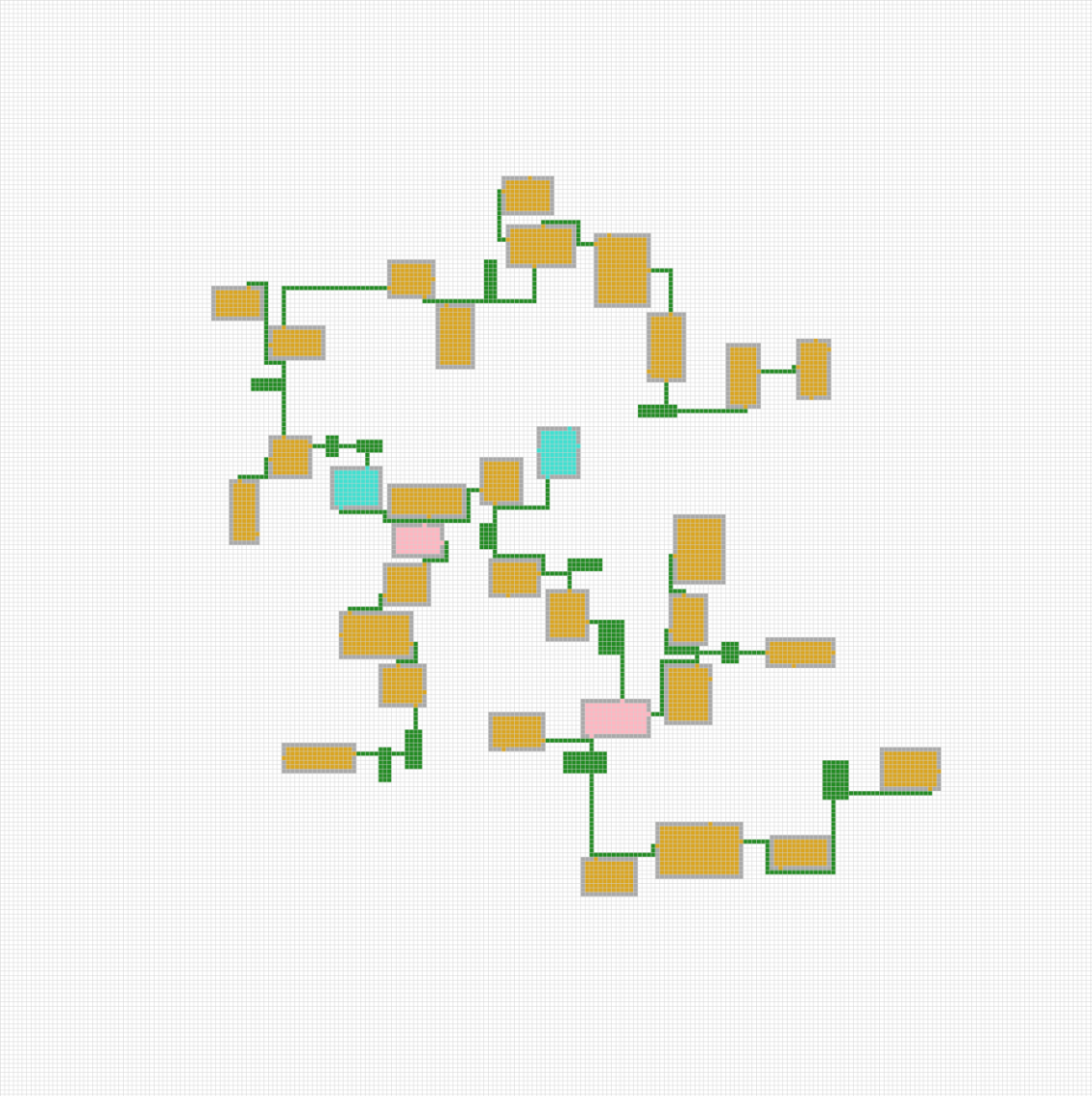


Figura 9 – Um outro nível gerado usando o gerador de mapas

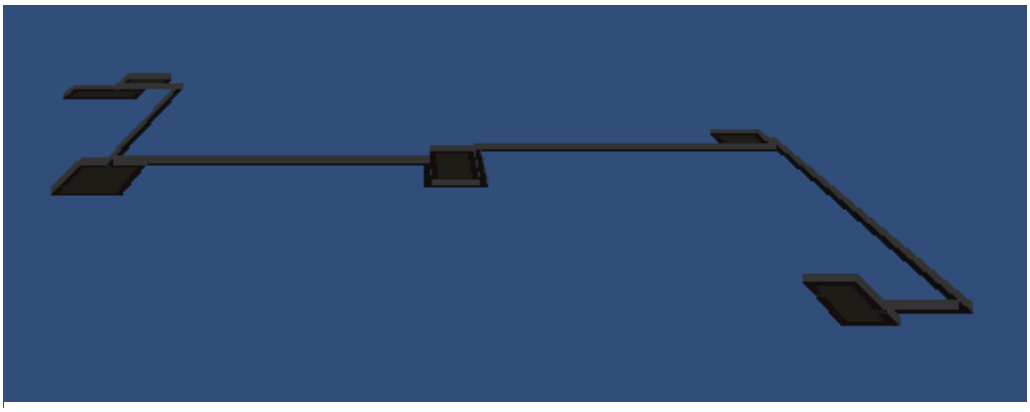


Figura 10 - Um nível visto de cima na versão mais antiga do Orion (paredes)

A informação de um nível é gerada a partir de um código fornecido no âmbito do projeto Orion [1], um algoritmo parecido com o de Prim [28] que gera um mapa do estilo dos jogos de Dungeon Crawler e ROGUE, com vários quartos, tendo informação de cada célula (se é chão, corredor, quarto, ou parede do quarto), tem informação de quantos quartos tem, e qual a área dos quartos, com o gerador tem de ser capaz de gerar um jogo em 3D, podia também ser usado outro algoritmo, só é preciso considerar a informação que é disponibilizada, sendo que alguma parte desse código está em grafos [12] [17], em que cria caminhos consoante "custos" que são atribuídos a cada tipo de célula [10].

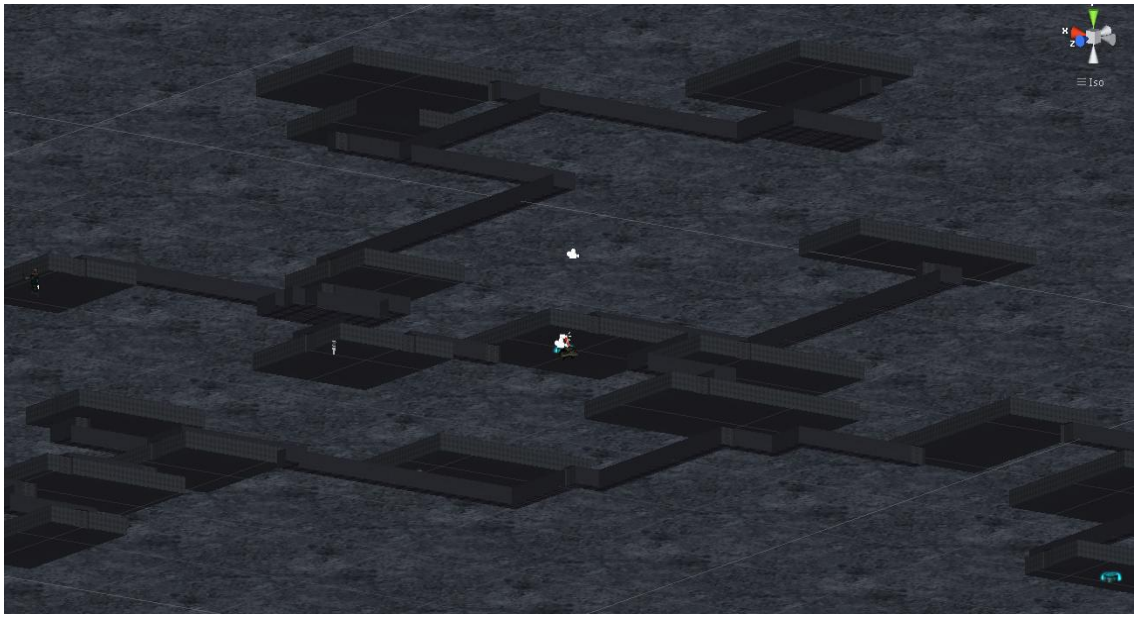


Figura 11 - Um nível visto de cima numa versão quase final(characters pieces report)

Para este tipo de jogo de Dungeon Crawler, o mais importante é saber cada célula, esse é o tipo básico, cada mapa é gerado normalmente com 250x250 células, normalmente estes mapas são 2D, com isto já é possível fazer qualquer tipo de jogo deste estilo, seja ele 2D ou 3D.

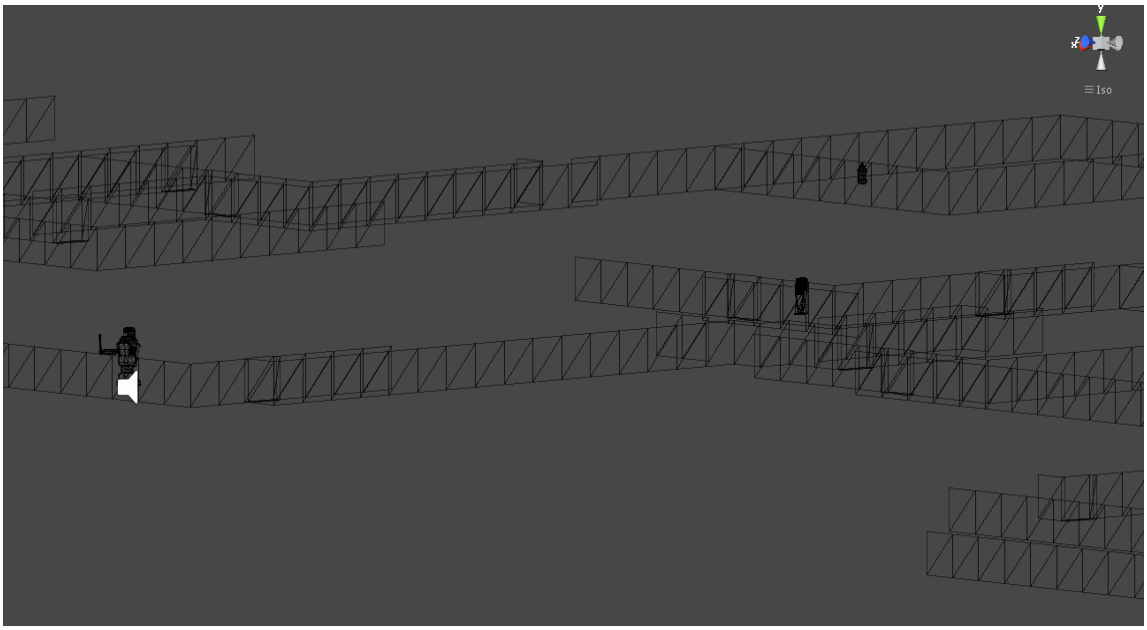


Figura 12 - Um nível visto por baixo

Do chão e das paredes só se vê a parte exterior, por exemplo o chão não é desenhado quando o jogador olha para o chão por baixo, isto permite alguma otimização de performance. Esta técnica é chamada Face Culling e os objetos gerados e instanciados são criados usando esta técnica. A ordem do desenho dos vectores das faces foi dos ponteiros do relógio (clockwise).

Com mais alguma informação do algoritmo que gera o mapa, por exemplo a área dos quartos, é mais eficaz colocar objetos por exemplo no centro de um quarto, mas também pode servir para ser mais fácil desenhar os quartos, por exemplo desenhar um mesh de quarto com essa área, poupando tempo do que desenhar célula a célula.

Figura 13 - Visão na terceira pessoa do jogo

Para o jogo ser 3D é preciso adicionar uma terceira dimensão para isso, podemos adicionar altura a todos os objetos, assim o jogo fica com gráficos 3D com uma jogabilidade 2D. Para ter uma jogabilidade 3D é possível adicionar vários níveis na vertical uns por cima de outros ficando como uma espécie de prédios (sendo um nível um andare), ou separados (lado a lado) dependendo do tipo de jogo. O personagem no início saltava (movimento vertical), e apesar de as colisões estarem feitas, como não havia nenhuma mecânica interessante a implementar com isso (podiam existir plataformas), mas como isso ficou excluído e essa habilidade retirada por se tratar de um jogo mais de exploração roguelike.

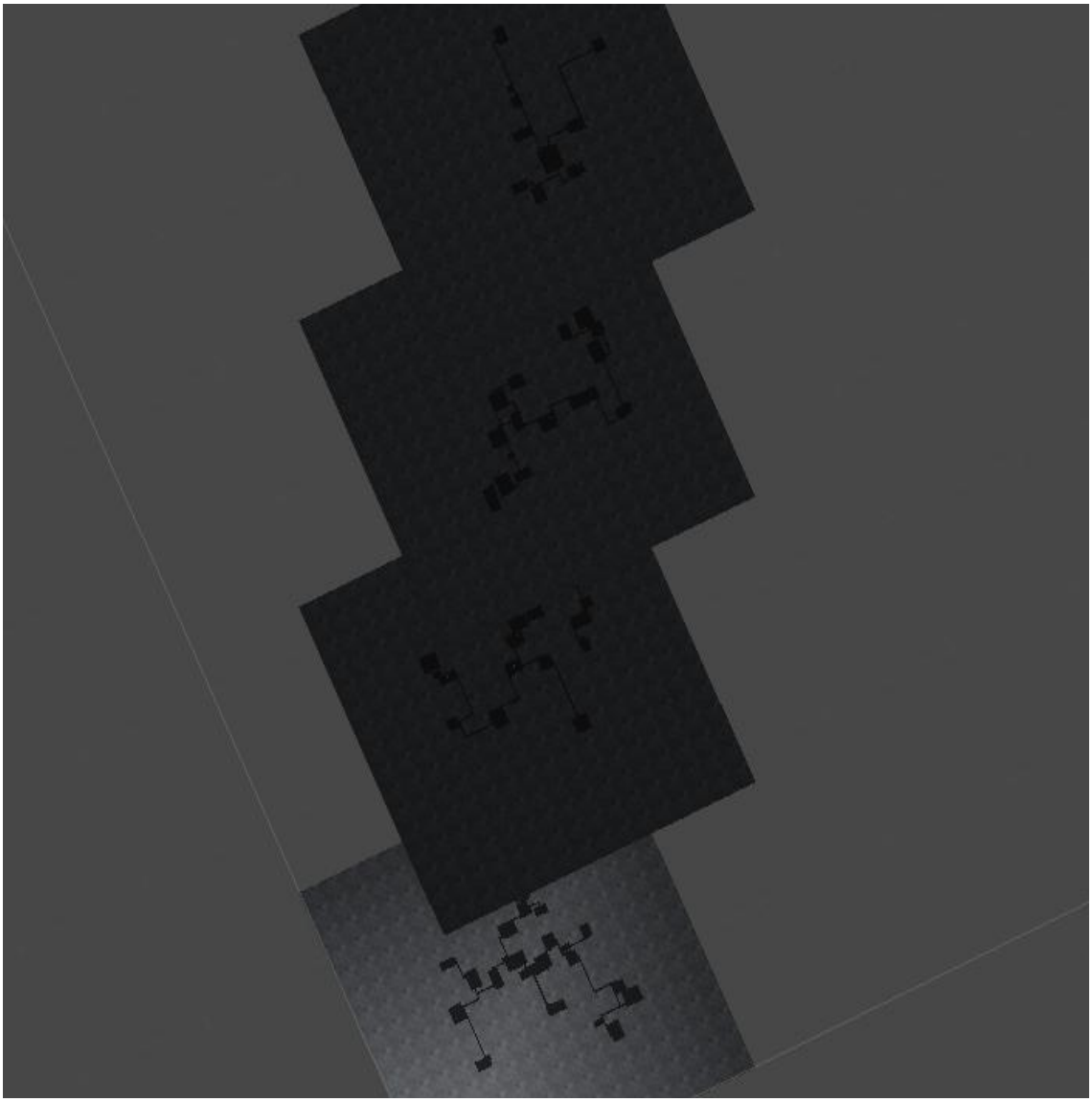


Figura 14 - Os vários níveis criados

Uma possível melhoria depois de ver como ficam os níveis é fazer com que cada nível seja desenhado apenas quando se está nele e não desenhar os outros, por questões de *performance*, os outros níveis podem ficar na memória permitindo ao jogador voltar ao nível anterior tal como ele era, não gerando um nível novo. Esta melhoria só não foi feita por questão de tempo, mas fica aqui referida como uma técnica de otimização em qualquer jogo PCG.

Cada nível passa a outro nível através do teleporte, tendo como objetivo do jogo fazer todos os exercícios, é necessário chegar a todos os níveis e para a narrativa é necessário apanhar peças para montar a nave, cada nível tem ainda mecânicas adicionais geradas ou descritas no XML da narrativa como destruir um robô, sendo estas mecânicas importantes para o jogo [19] e o seu divertimento [20].

3.3.2 Instanciação de objetos

Os objetos são instanciados recorrendo a informação que vem do XML onde se lê quantos objetos tem, dependendo do tipo de jogo, normalmente é melhor ler primeiro e saber quantos objetos vão ter de ser criados, isto no caso de o objeto ter de ocupar um lugar específico, se isso não acontecer ou não for possível, o que se deve fazer é ir preenchendo nos espaços possíveis.

No caso do Orion o espaço apropriado para grande parte dos objetos era por exemplo no meio de um qualquer quarto, e para isto convém ser fácil encontrar um quarto, o que pelas células é possível, sendo que a célula que tem o tipo "chão do quarto" está no quarto, mas se quiser ser instanciada num determinado local do quarto como no meio, é necessário algo mais que isso, existem várias formas de o fazer, no caso do Orion existe uma função do gerador do mapa que permite saber a área do quarto, e a partir daí é fácil colocar o objeto em qualquer posição dentro de uns certos parâmetros [22].

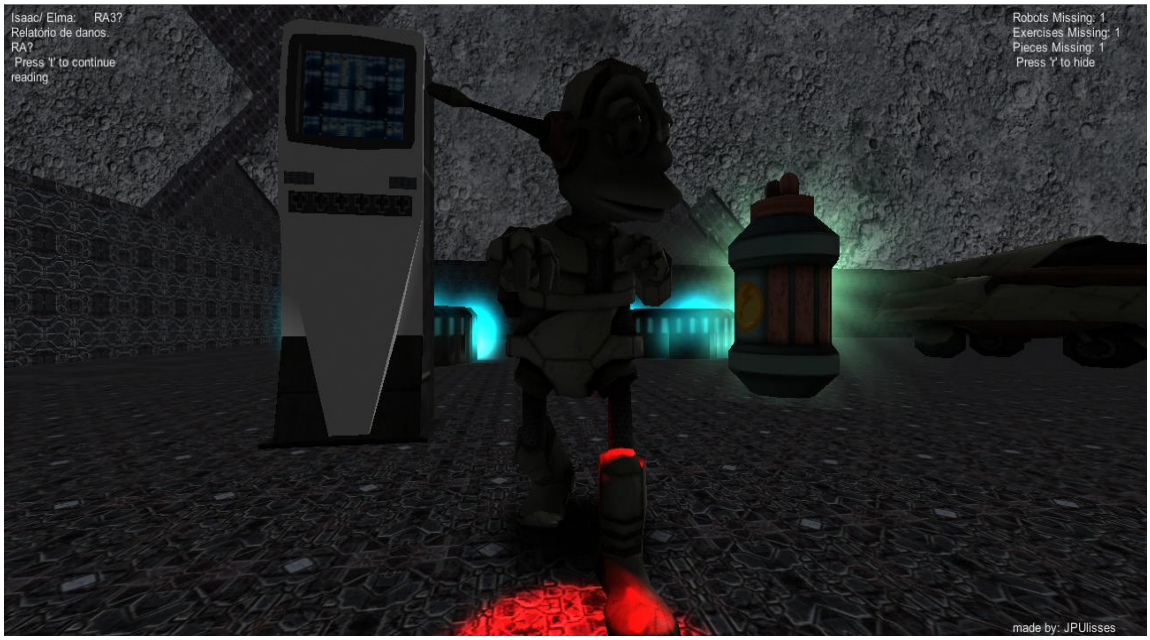


Figura 15 - Vários objetos num quarto

3.4 Desenvolvimento do jogo sério

Houve preocupação em certificar que o jogo sério era jogável e divertido. Para tal foi tido em conta o espaço que o jogador tem que correr, espaço gerado e a velocidade do personagem que o jogador controla entre outras variáveis para ser mais divertido. O jogo poderia ter em conta a disposição do jogador como em [30], mas como se trata de um jogo sério é complicado limitar o conteúdo gerado para que o jogador não faça batota, o mecanismo dinâmico reside mais nas capacidades do jogador, para isto o docente deve criar vários caminhos para o jogo, uns fáceis, outros difíceis, caminhos que o jogo pode indicar conforme as capacidades do jogador em resolver os exercícios e o caminho mais apropriado.

O jogo dá indicações ao jogador para saber onde deve ir e o que procurar. O jogador quando luta com um robô pode levar socos dele, mas nunca chega a ser derrotado, nem o jogador morre, isto porque se morresse podia ir para outro local para reaparecer e teria uma gestão de vidas podendo chegar ao fim do jogo antes de completar os exercícios, o que não era pretendido, então o jogador é como se fosse imortal, se levar socos vai a voar para trás perdendo algum tempo, mas pode sempre voltar a lutar contra o robô e tentar desviar-se melhor dos seus ataques.

Existem objetos e exercícios opcionais para ser mais apelativo ao jogador encontrar caminhos diferentes, isto pode também estar associado a caminhos mais fáceis, informação que vem do XML.

Multiplayer ou monitorização

O jogo poderia ser jogado num modo para vários jogadores, pois todos os objetos e componentes independentes funcionam de forma independente podendo facilmente ser instanciados e controlados por outras máquinas tendo um resultado igual em todos os computadores, sendo apenas necessário comunicar alguma informação exclusiva como a posição do jogador e as suas ações.

O mapa tinha de ser previamente escolhido para os jogadores jogarem, bastava comunicarem entre si e decidirem em que mapa vão jogar e quem o gera e enviar aos outros jogadores, além disto atualizar as posições dos *players* quando se movem e quando algum evento dos objetos é ativado, por exemplo quando uma porta abre ou um robô inimigo é destruído, criando assim um modo de jogo novo muito rapidamente

O único senão é que podia ser considerado um elemento distrativo, visto que se trata de um jogo sério, mesmo assim ainda pode ser aproveitado como monitorização ou algo do género, visto que é relativamente fácil comunicar quais são os objetos e eventos que se modificam devido à organização do código e à relação entre as variáveis no jogo (como por exemplo a posição relativa).

Conclusões da narrativa e XML

Como é possível observar, existem diferentes tipos de informação, existem tipos de dados como as células que são a unidade básica de um mapa, que tem um certo tamanho, textura, entre outras variáveis que lhe pertencem, e existem variáveis que englobam essas, como quarto e mapa entre outras, esta informação pode parecer redundante, mas podem dar jeito particular nas funções de render ou de inteligência artificial do jogo, tornando a pesquisa mais rápida e poupando tempo de processamento, a organização destes dados numa linguagem de alto nível é também muito importante pois vai facilitar o acesso à informação.

Aliando a uma linguagem orientada a objetos permite instanciar qualquer tipo de objeto, num dado sitio dependendo do algoritmo e das variáveis do jogo, isto pode ir mais além com variáveis externas, que podem controlar desde os próprios gráficos (texturas, número de vértices, modelo), como som, narrativa, até objetos externos ao programa (deste que tenham uma semântica válida com o Unity).

A seguir será continuada a demonstração e a importância destes pontos referidos e sua utilidade para qualquer jogo.

3.5 Opções de desenvolvimento

Existem algumas questões relativamente à melhor forma de gerar conteúdo, grande parte delas está dependente de múltiplos fatores que têm que ser usados na geração e além disto têm de estar todos de acordo para criar uma maior harmonia (quanto possível) permitindo ao jogo ser gerado com uma melhor otimização e *performance* máxima.

Memória vs. Processamento

Quando se gera um jogo, há sempre a questão de ele ser gerado enquanto o jogador está a jogar ou carregar um nível pré gerado, no caso do Orion poder-se-ia fazer qualquer um destes, aqui é difícil decidir qual é melhor pois depende do jogo pretendido, tendo estes tipos vantagens e desvantagens que podem ser favoráveis ou desfavoráveis dependendo do tipo de jogo.

Em jogos em que o mundo do jogo é gerado e que o jogador pode optar por encerrar o jogo e iniciar outro (gerando assim um novo mundo), o computador que gera tem que processar essa informação o mais rapidamente possível, e depois guarda o necessário na memória, eliminando todo o que não precisa, incluindo eliminar toda a informação no fim do processo, isto é importante pois dependendo do garbage collector que está a ser utilizado, pode ser necessário alguma gestão adicional.

O outro caso, são jogos que vêm pré gerados, como na maior parte dos jogos que não são PCG em que os níveis são estáticos, com um melhor caso possível ou mais agradável para o jogador, e compactam essa informação que depois é lida, mas o mundo do jogo nesses jogos é sempre o mesmo. Em jogos PCG é possível existirem níveis estáticos, sendo o melhor caso possível para agradar o jogador ou por questões de testes.

Existem também casos intermédios, em que os objetos que o jogador recebe, encontra ou a sua localização é aleatórias num mapa pré gerado, existem também nestes jogos partes ou mecânicas que são geradas procedimentalmente

Em jogos mais antigos, é comum o caso dos jogos pré gerados, talvez porque os processadores dos computadores que as pessoas tinham não fossem poderosos o suficiente, e também temiam o impacto dos "loading screens" se fossem demasiado longos, e em jogos em que a narrativa é fixa também se nota muito este problema.

Em jogos de geração de grande escala foram feitos testes em [6] que também tratam de questões relativas a este tema num outro contexto.

O termo "sandbox" muitas vezes usado para os jogos tem o mundo gerado, este mundo é aberto e normalmente quando o jogador chega a uma das extremidades geradas, gera mais conteúdo para o jogador explorar, um dos casos mais conhecidos é o famoso jogo Minecraft em 2009 [5].

Soluções

É difícil dar uma solução geral, por existirem muitas variáveis em questão, para começar o tipo de jogo e os seus objetivos, esse deve ser o primeiro ponto para decidir que partes é que vão ser geradas procedimentalmente, depois de este passo estar resolvido é que vamos abordar a questão se é melhor pré-gerar o conteúdo ou gerar enquanto o jogador está a jogar.

Os problemas para o jogador aqui são, no caso de o jogador não gostar do conteúdo, se for pré-gerado pouco ou nada pode fazer, se for gerado, ele pode tentar pedir para gerar outro, se bem que em conteúdo pré-gerado há um maior controlo de qualidade do conteúdo tornando isso mais raro, mas pode acontecer, e a solução para a segunda parte, pode ser considerada má se o jogador considerar batota gerar até aparece um conteúdo mais fácil que outro, claro que isto pode ser controlado e limitado, sendo estes problemas resolvidos no código do jogo ou com testes sobre qualidade do conteúdo.

Porém ainda não é certo qual a solução mais rápida ou eficaz, ou seja, ela depende no fundo do *hardware* do computador que está a correr o jogo, muitos jogos hoje em dia, têm várias opções gráficas que permitem otimizar a *performance* do jogo, mas nem sempre são perfeitas e tentar descartar estas questões para o jogador, ou para o *hardware*, quando o próprio código pode ajudar e muito na *performance*. Este se for capaz de identificar o *hardware*, pode calcular o melhor caminho, que não tem necessariamente de ser o mais rápido, mas sim o melhor em termos de performance, uma boa métrica para ser usada, são os fps (frames per second), se o jogo tiver que processar demasiada informação os fps baixam, ou se o jogador tiver constantes paragens com ecrãs para carregar mais conteúdo ou gerar, mas o mesmo pode também acontecer se o jogador for todo gerado por completo, ou pré-gerado ocupando um espaço demasiado grande na memória para o computador conseguir mostrar com fps fluidos.

Em conclusão dependendo de cada computador, o jogo gerado pode ser otimizado para ele, conforme o poder das memórias e do processador do computador, há que ter em conta também o processador da placa gráfica os componentes gráficos, como por exemplo apenas desenhar o nível em que o jogador está actuamelmente (ou mais dependendo da placa gráfica), em memória apenas fica a área imediatamente mais próxima ou o nível seguinte (ou mais dependendo do espaço da memoria), e a área a seguir pode ser gerada procedimentalmente (em que o jogador espera consoante a velocidade do processador, tendo esta parte de ser o mais disfarçada possível com o menor custo de fps). Alguns jogos usam esta técnica, mas deixam muito à responsabilidade do jogador e do *hardware* correr o jogo de forma certa.

O tamanho do mundo do jogo, a complexidade do mundo e a localização dos objetos se for gerada, têm uma correlação inversa com o que vai ser desenhado pela placa gráfica, ou seja dependendo do *hardware*, a memória e o processador devem aliviar o trabalho da placa gráfica para que possa ter um nível agradável de fps para o jogador,

Para o jogo Orion isto foi tido em conta, porém para o *hardware* do computador utilizado em especifico, ou seja para outros *hardwares* era necessário alterar alguns valores, como componentes gráficos (exemplo qualidade dos modelos, texturas), quantos elementos ficam na memória entre outras questões para ficar adequado, e claro isto vária de jogo para jogo, de elemento para elemento e de computador para computador, mas a solução teórica e as métricas são as mesmas, só muda a precisão dos valores.

Comunicação entre linguagens de programação, elementos externos, exceções e limitações de conteúdo

Como foi referido anteriormente o jogo Orion está programado em C# e usa alguns elementos criados no tutorial do Unity3D, esses estão programados em Javascript, a instanciação de objetos e elementos externos ao jogo é importante, pois permite criar um jogo mais diversificado e com menos esforço, por exemplo uma equipa pode tratar da parte de gerar o mundo, enquanto outra está a fazer um elemento especifico para ficar num determinado espaço nesse mundo, seja ele objeto, modelo, textura, som ou outros.

Contudo existe o problema que se um objeto mais complexo ou com vários elementos, estiver pronto a ser compilado em outra linguagem diferente da que o mundo gerado é compilado, cria problemas de compatibilidade, requerendo uma conversão ou outro objeto, contudo, existe uma solução melhor, que é recorrendo ao Unityscript, estes objetos podem ser instanciados independentemente da sua linguagem, contudo só assim, o objeto não poderia comunicar com o mundo, estaria apenas ali presente até ser destruído.

A solução para comunicarem está também presente no Unityscript, podendo por exemplo chamar um método que evoca uma função nesse objeto e pode passar-lhe todos os seus valores, nessa nova função na mesma linguagem do mundo gerado e é fácil aceder às suas variáveis, permitindo assim a comunicação.

Por exemplo quando o robô (objeto feito em javascript) é destruído, chama a função Die() que por sua vez chama SendMessage() que vai correr uma função em C# podendo assim comunicar com tudo o que está no jogo, sendo que a estrutura e lógica do jogo estão feitas em C# podendo assim diminuir o número de robôs existentes no mundo, depois então o objeto é destruído.

```
function Die ()
{
SendMessage ("EnemyReduce");
Destroy (gameObject);
```

Contudo pode acontecer o gerador ficar à espera de uma determinada informação que pode não chegar ou não existir por opção (de quem criou o ficheiro XML), e o gerador deve ser capaz de continuar a gerar sem essa informação, por exemplo com “try catch”, em que se uma lista estiver vazia, não a lê, é importante gerir e salvaguardar estas pequenas partes pois à medida que o código vai ficando mais complexo vão começar a aparecer erros, e desta forma eles são controlados.

Outro exemplo pode ser se não existir nenhum robô, poder criar um, ou nos switch cases, correr sempre um valor por definição, ou nos valores nullable, se for encontrado um valor inesperado, correr um valor por definição com o método dos nullable ".GetValueOrDefault (0) " (em que 0 é o valor por definição).

Outro problema relativo ao conteúdo é quando o mundo é limitado ou o conteúdo vai ser criado num espaço que não existe, estas variáveis têm de ser limitadas para ficarem dentro do espaço útil, e se ainda acontecer um erro, deve correr um valor seguro por definição, como normalmente se faz com divisões por zero.

Conclusões da integração entre linguagens

Como se pode concluir, estas considerações devem ser tidas em conta em todos os jogos, especialmente os jogos que usam PCG, com a acrescida complexidade é comum ficarem fora de controlo, com algo a limita-los é mais fácil controlá-los, principalmente quando se trata de variáveis relativas, mais difíceis de analisar no código.

A comunicação entre várias linguagens permite aos criadores desenvolverem conteúdo independentemente da linguagem, poupando tempo valioso e permitindo que estes participem até mais tarde no desenvolvimento do jogo, podendo por exemplo criar arte para o jogo em fases finais do projeto.

Figura 16 - Arte conceptual do personagem principal (desenhado por Enrique Kato).[3]

Testes

O jogo foi realizado em menos de dois meses, por isso já se sabia que havia pouco tempo para testar, pelo que foi adotada uma forma de testes unitários e com os limites mencionados em cima, foi fácil reduzir o número de problemas, testando o jogo com mais facilidade

Foram realizados vários testes, que se podem dividir em dois grandes tipos, os testes internos que foram feitos ao longo do projeto e depois testes adicionais no fim, com registo de valores, isto quer para encontrar problemas quer para os solucionar, mas também por questões de *performance*, melhorias e também para apresentar resultados. O outro tipo de teste foi, a meio do projeto e no fim, criar executáveis, das versões do jogo e pedir a pessoas próximas, com diferentes computadores para testarem. A participação do autor desta dissertação neste teste também foi feita com outros computadores que possuía, o objetivo deste teste não era tanto ver a reação do jogador, mas sim ver se o jogo corria sem qualquer problema, e a velocidade de geração era suficientemente boa para as pessoas não acharem isso um problema, o executável do Unity permitiu alguma qualidade neste teste pois para vários computadores foi sempre possível correr o jogo, pois o executável permite várias qualidades e o Unity permite criar executáveis para várias plataformas, android, web, Macintosh e Windows (32 ou 64 bits) entre outras, a única testada foi em Windows devido à maior parte das pessoas só usar Windows e foram esses os executáveis criados.

Foi testado o jogo em termos de fps para ser o mais agradável possível ao jogador e foram alterados vários parâmetros como os modelos, as texturas, quantos quartos e níveis o jogo deveria ter, uma questão interessante foi que o algoritmo que gerava o mapa, e criava caminhos através de "custos", como um algoritmo de pesquisa, ou neste caso como um algoritmo de Prim[28], é mais eficiente quanto mais quartos existirem, e de forma inversa quanto mais quartos existirem mais pesado vai ser à placa gráfica desenhá-los, por isso teve de ser encontrado um meio-termo, usando fps como métrica novamente.

O exemplo do jogo será incluído em anexo, para que qualquer pessoa que queira testar, ou simplesmente jogar, poder fazê-lo.

Nome da versão / Tipo da versão	Data	Média do tempo da geração do mapa (segundos)	Média do tempo da geração dos restantes elementos (segundos)	fps	Número de Quartos	Número máximo de Níveis
Inicial	2 de maio de 2014					
Paredes	6 de maio de 2014	30.6158935	30.712923	67.25	10	1
Paredes + textura 2	8 de maio de 2014	17.5310204	17.6952286	62	10	1
player spawn 2	9 de maio de 2014	13.9072622	14.064323	71.4	10	1
portas a abrir	15 de maio de 2014	26.052906	26.169656	68.2	10	1
player e enemy spawn	16 de maio de 2014	15.8457118	15.9648558	67.6	10	1

Tabela 3 – Versões Inicial com início em 2 de Maio de 2014

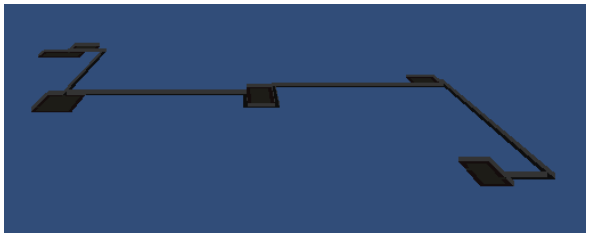


Figura 17 – Imagem versão Paredes

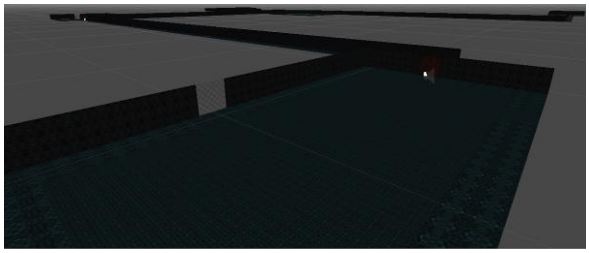


Figura 18 – Imagem versão Paredes + textura 2

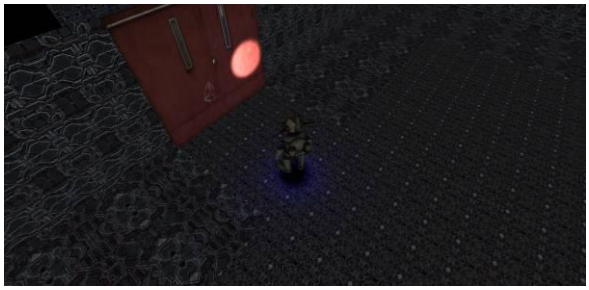


Figura 19 - Imagem versão player spawn 2

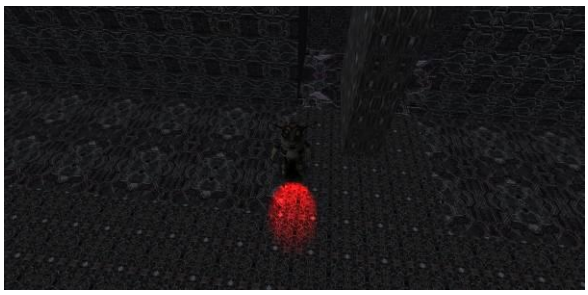


Figura 20 - Imagem versão portas a abrir

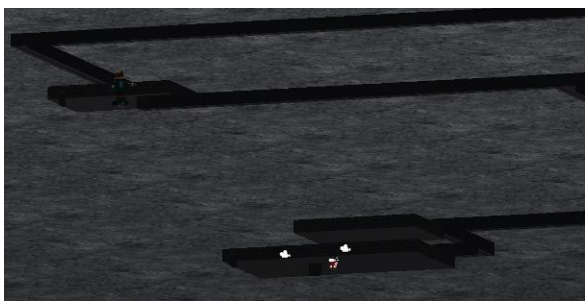


Figura 21 - Imagem versão player e enemy spawn

Nome da versão / Tipo da versão	Data	Média do tempo da geração do mapa (segundos)	Média do tempo da geração dos restantes elementos (segundos)	fps	Número de Quartos	Número máximo de Níveis
Níveis	24 de maio de 2014					
níveis camera teleport	24 de maio de 2014	91.189938	92.274874	55.6	15	7
teleport nos sítios	30 de maio de 2014	89.88984	90.89683	52.2	15	7
robots nos sítios	1 de junho de 2014	83.304562	84.738922	44.8	30	7

Tabela 4 – Versões Níveis com início em 24 de maio de 2014

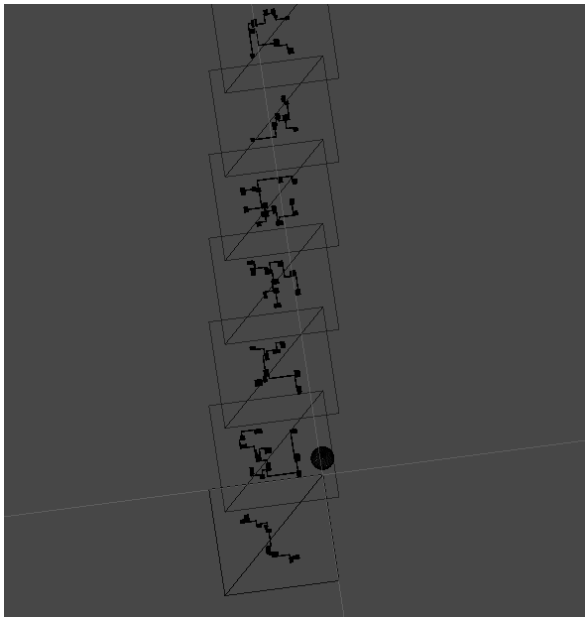


Figura 22 - Imagem versão níveis camera teleport

Figura 23 - Imagem versão teleport nos sítios

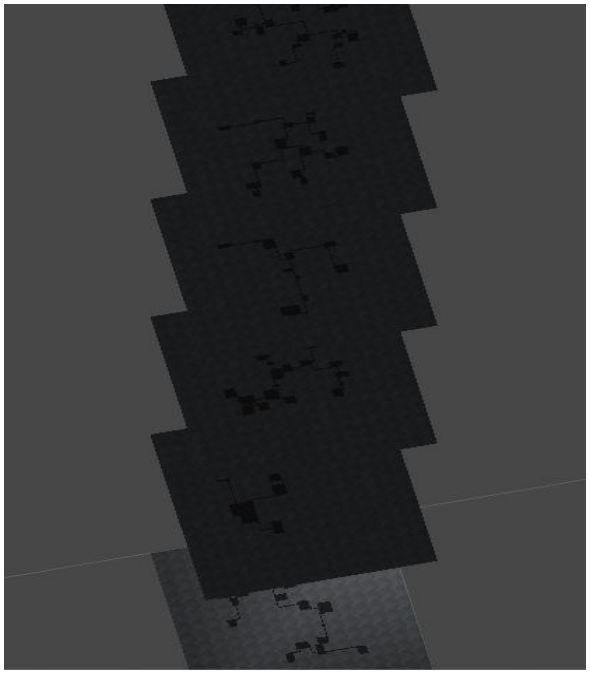


Figura 24 - Imagem versão robots nos sítios

Nome da versão / Tipo da versão	Data	Média do tempo da geração do mapa (segundos)	Média do tempo da geração dos restantes elementos (segundos)	fps	Número de Quartos	Número máximo de Níveis
Narrativa/XML	1 de junho de 2014					
xml primeira	2 de junho de 2014	85.211342	86.778038	46.6	30	7
xml leitura	11 de junho de 2014	67.13889	68.360512	45	30	6
robots	17 de junho de 2014	74.444592	75.695234	53.4	30	6
exercise	18 de junho de 2014	61.098546	62.279444	52	30	6
characters pieces report	19 de junho de 2014	63.462746	64.762914	44.4	30	6

Tabela 5 – Versões Narrativa/XML com início em 1 de Junho de 2014

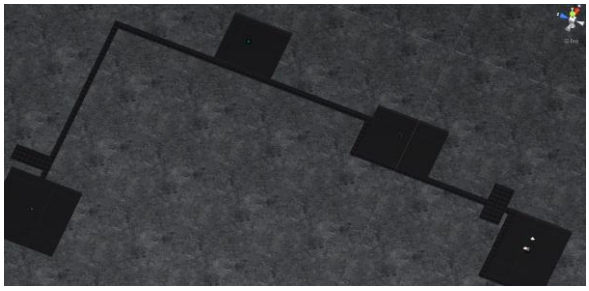


Figura 27 - Imagem versão xml primeira

Figura 25 - Imagem versão xml leitura

Figura 28 - Imagem versão robots

Figura 29 - Imagem versão exercise

Figura 26 - Imagem versão characters pieces report

Estes testes foram realizados num computador pessoal, as suas características e outros detalhes sobre os testes podem ser consultados no anexo.

Conclusões dos testes e da experiência do projeto

Com estes testes foi possível melhorar a *performance* do jogo e verificar que o projeto consegue correr em vários computadores, o conceito do jogo e os seus elementos tinham de ser testados de uma forma diferente mais pormenorizada, mas como não houve necessidade, optou-se por acabar os testes.

Jogos que utilizam métodos PCG devem:

- Permitir a instanciação dos objetos de diferentes linguagens de programação e comunicação entre eles garantindo uma maior diversificação de conteúdo e permitir aos artistas modificar conteúdo em fases finais de desenvolvimento.

- Ter os seus objetos bem referenciados para serem mais facilmente substituídos.

- Utilização de variáveis relativas o mais complexas possível e se necessário recorrer a limites e exceções para evitar casos impossíveis. (por exemplo o caso dos valores por definição e variáveis nullable).

- O conteúdo deve ser gerado ao máximo de acordo com o *hardware*, podendo ser pré-gerado e o nível ser mais estático, ou gerado em tempo real e mais dinâmico, garantindo uma jogabilidade o mais interessante possível e fluído (os fps (frames por segundo) são uma possível métrica para ver se um jogo está a correr sem falhas).

Capítulo 5

Conclusões e Trabalho Futuro

4.1 Conclusões

A geração usando documentos ou objetos externos permite que haja uma maior colaboração entre diferentes áreas trabalhando ao mesmo tempo em tarefas diferentes e que a sua integração seja mais fácil, permitindo que, por exemplo, artistas possam trabalhar numa parte final do projeto, sendo isto benéfico para o produto final [18].

Quanto mais relações existirem entre os vários conteúdos, maior será a qualidade do jogo gerado, mais complexo será, mas com boas escolhas de game *design* o jogo será muito satisfatório, junto a um código gerador será possível gerar o jogo a partir desse documento.

Este projeto permitiu demonstrar que é possível editar ficheiros XML e criar em pouco tempo novas versões do jogo. Isto pode ser testado no jogo em anexo, editando o XML que está dentro das pastas do jogo. Sendo este conteúdo editável e facilmente mudado criando rapidamente novas versões de jogo uma vantagem relativamente única nos dias de hoje para jogos sérios.

Cada vez mais os jogos que usam métodos PCG exigem pessoas de áreas multidisciplinares, em que além de programarem têm de ter um conhecimento relacionado com o conteúdo a ser gerado, onde só o conhecimento de uma das áreas não chega para um bom resultado.

4.2 Trabalho Futuro

Ainda falta reforçar e unir as ligações entre várias áreas que podem beneficiar o desenvolvimento de um jogo, porém, com estas soluções, estamos cada vez mais perto, sendo que, cada uma destas deve estar relacionada com as outras todas, além da ponte que os une, que normalmente será o gerador, que lê todas estas informações.

O som seria um próximo passo, por exemplo gerando uma narrativa o som ambiente será mais assustador com sons graves, enquanto uma bela paisagem podia gerar sons que se parecessem com pássaros, enriquecendo o jogo e a sua qualidade.

Conteúdo gráfico aliado à performance pode ser explorado para todos os tipos de jogos, este conteúdo pode ainda permitir aos jogadores algum dinamismo se associado também ao conteúdo gerado pelos jogadores criando uma espécie de conteúdo perfeito onde tudo é considerado, porem para isto é necessário ter em conta todas as limitações, que podem ser feitas por uma comunidade.

Todo o conteúdo bem estruturado pode ser o combustível para gerar jogos a partir de um documento, podendo este ser um novo motor de jogo, quanto mais conteúdo e melhor estiverem detalhadas as suas relações, mais complexo e completo será o jogo gerado.

Referências

[1] A. Coelho, E. Kato, J. Xavier, R. Gonçalves, Serious Game for Introductory Programming

[2] R. Ferreira Gonçalves, Jogo Digital para o Ensino dos Fundamentos da Programação, 17 de Junho 2011.

[3] E. Romo, Game design for a serious game to help learn programming, 29 de Julho 2011.

[4] Acornsoft, "Elite," 1988.

[5] M. Persson e J. Bergensten, "Minecraft" em http://www.minecraft.net, 2009.

[6] E. Carpenter, Procedural Generation of Large Scale Gameworlds, Setembro 2011.

[7] N. Gillian, S. O'Modhrain, e G. Essl. Scratch-Off: A gesture based mobile music game with tactile feedback. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Pittsburgh, Junho 2009.

[8] A. Jordan, D. Scheftelowitsch, J. Lahni, J. Hartwecker, M. Kuchem, M. Walter-Huber, N. Vortmeier, T. Delbrügger, Ü. Güler, I. Vatolkin, e M. Preuss. Beatthebeat – music-based procedural content generation in a mobile game, em *Computational Intelligence and Games (CIG) 2012 IEEE Conference*.

[9] J. Togelius, A. Champandard, P. Lanzi, M. Mateas, A. Paiva, M. Preuss, e K. Stanley, Procedural Content Generation: Goals, Challenges and Actionable Steps.

[10] J. Togelius, G. Yannakakis, K. Stanley, e C. Browne, Search-Based Procedural Content Generation: A Taxonomy and Survey 2011.

[11] G. Yannakakis, J. Togelius, Experience-Driven Procedural Content Generation, Julho-Setembro 2011.

[12] F. Luca, CASTLE:Map Generation and Navigation 2012.

[13] K. Perlin. Improving noise, 2002.

[14] A. Lindenmayer P. Prusinkiewicz. The Algorithmic Beauty of Plants. Springer-Verlag, segunda edição, 1996.

[15] A. Coelho, Sistemas L, 2008.

[16] P. Oppenheimer, "Real time design and animation of fractal plants and trees," 1986.

[17] D. Adams Automatic Generation of Dungeons for Computer Games, 2002.

[18] M. Hendrikx, S.Meijer, J. Velden e A. Iosup, Procedural content generation for games: a survey, Fevereiro 2013.

[19] J. Dormans e S. Bakkes, Generating Missions and Spaces for Adaptable Play Experiences Setembro 2011.

[20] J. Huizinga, Homo Ludens: o jogo como elemento de cultura, 2001.

[21] J. Togelius, T. Justinussen, A. Hartzen, Compositional procedural content generation.

[22] V. Valtchanov, J. Alexander Brown, Evolving Dungeon Crawler Levels With Relative Placement.

[23] M. Fahey, "It's Not The Size Of The Game World, But How You Use It," no. 25/6/2011. 2010.

[24] Bethesda Softworks LLC, "The Elder Scrolls Official Site," no. 9/7/2011. 2011.

[25] E. Matthews, B. Malloy, Procedural Generation of Story-Driven Maps, 2011.

[26] T. Susi, M. Johannesson, P. Backlund, Serious Games – An Overview 2007 .

[27] w3c XML validator, http://www.w3schools.com/xml/xml_validator.asp

[28] Prim, R. C., "Shortest Connection Networks and Some Generalizations", Novembro 1957.

[29] O. Cueilliez, http://www.svgopen.org/2011/papers/14-An_Original_Approach_to_Web_Game_Development_Using_SVG/

[30] A. Ferreira, "Geração procedimental de níveis de jogo com base no conceito de play-persona", 19 de Junho de 2012.

[31] Dungeon Crawl Stone Soup, gerador de mapas, https://crawl.develz.org/wiki/doku.php?id=dcss:brainstorm:dungeon:layouts , https://crawl.develz.org/wiki/doku.php?id=dcss:help:maps:advanced

[32] Wizards of the Coast, gerador de mapas, http://www.wizards.com/dnd/mapper/launcher.htm

[33] Random Dungeon Generator, http://donjon.bin.sh/fantasy/dungeon/

[34] Unity Script, http://docs.unity3d.com/ScriptReference/ , http://wiki.unity3d.com/index.php/UnityScript_versus_JavaScrip

[35] A. Liapis,G. Yannakakis, J. Togelius, Towards a Generic Method of Evaluating Game Levels, 2013.

[36] Huang, Timothy, Strategy game programming projects. Small 4, no. Maio 2001: 205-213.

[37] Berlin Interpretation, http://www.roguebasin.com/index.php?title=Berlin_Interpretation, 2008.

[38] Dungeon Crawl Stone Soup, http://crawl.develz.org/wordpress/

[39] I. Brito, “Narrativa para Video Jogos Sérios” Julho 2014

Anexo A

Teste ao Jogo Orion

Informações adicionais relativamente aos testes efetuados.

6.1 Especificações do computador onde foi testado o jogo Orion

Operating System: Windows 8.1 Pro 64-bit (6.3, Build 9600) (9600.winblue_gdr.131030-1505)
Language: Portuguese (Regional Setting: Portuguese)
System Manufacturer: Hewlett-Packard
System Model: HP G62 Notebook PC
Processor: Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz (4 CPUs), ~2.5GHz
Memory: 4096MB RAM
Available OS Memory: 3894MB RAM
Page File: 2332MB used, 5529MB available
DirectX Version: DirectX 11
User DPI Setting: Using System DPI
System DPI Setting: 96 DPI (100 percent)
Card name: ATI Mobility Radeon HD 5470
Manufacturer: ATI Technologies Inc.
Chip type: ATI display adapter (0x68E0)
DAC type: Internal DAC(400MHz)
Display Memory: 2806 MB
Dedicated Memory: 1014 MB
Shared Memory: 1792 MB

6.2 Versões e testes do jogo Orion

Inicial

Paredes

17.57734
17.64308
~69fps

9.228874
9.339652
~67fps

38.19504
38.31151
~66 fps

57.46232
57.55745
~67 fps

10 Rooms
1 Level

Média de tempo para a geração do mapa: 30.6158935 segundos
Média de tempo para a geração dos restantes elementos:30.712923 segundos
Média de fps:~67.25fps

Paredes + textura 2

9.042758
9.257684
~63fps

52.17617
52.34552
~60 fps

11.0135
11.15947
~61 fps

2.123404
2.209549
~65fps

13.29927
13.50392
~61fps

10 Rooms
1 Level

Média de tempo para a geração do mapa: 17.5310204 segundos
Média de tempo para a geração dos restantes elementos:17.6952286 segundos
Média de fps:~62fps

Player spawn 2

3.371915
3.548325
~75fps

15.44031
15.59399
~73fps

35.24043
35.40607
~69fps

6.879983
7.014064
~74fps

8.603673
8.759166
~66fps

10 Rooms
1 Level

Média de tempo para a geração do mapa: 13.9072622 segundos
Média de tempo para a geração dos restantes elementos:14.064323 segundos
Média de fps:~71.4fps

Portas a abrir

23.73679
23.8187
~69fps

24.84692
24.99519
~70fps

20.88905
21.0256
~67fps

38.74025
38.82305
~68fps

22.05152
22.18574
~67fps

10 Rooms
1 Level

Média de tempo para a geração do mapa: 26.052906 segundos
Média de tempo para a geração dos restantes elementos:26.169656 segundos

Média de fps:~68.2fps

Player e enemy spawn

27.78548
27.92592
~67fps

4.515911
4.614377
~70fps

23.21848
23.33044
~68fps

14.29348
14.42704
~67

9.415208
9.526502
~66

10 Rooms
1 Level

Média de tempo para a geração do mapa: 15.8457118 segundos
Média de tempo para a geração dos restantes elementos:15.9648558 segundos
Média de fps:~67.6fps

Níveis

Níveis camera teleport

112.3842
113.6167
~55fps

67.19243
68.3326
~65fps

85.78008
86.74933
~55fps

112.6502
113.7169
~52fps

77.94278
78.95884
~51fps

15 Rooms
7 Levels

Média de tempo para a geração do mapa: 91.189938 segundos
Média de tempo para a geração dos restantes elementos:92.274874 segundos
Média de fps:~55.6fps

Teleport nos sítios

79.44422
80.59148
~50fps

115.4691
116.4325
~55fps

104.7607
105.6625
~52fps

67.07597

68.07958
~53fps

82.69921
83.71809
~51fps

15 Rooms
7 levels

Média de tempo para a geração do mapa: 89.88984 segundos
Média de tempo para a geração dos restantes elementos:90.89683 segundos
Média de fps:~52.2fps

Robots nos sítios

72.73048
74.17681
~51fps

88.24664
89.63896
~40fps

108.3761
109.9822
~39fps

66.37989
67.95343
~42fps

80.7897
81.94321
~52fps

30 Rooms
7 levels

Média de tempo para a geração do mapa: 83.304562 segundos
Média de tempo para a geração dos restantes elementos:84.738922 segundos
Média de fps:~44.8fps

XML

xml primeira

83.00764
84.73807
~43fps

74.39676
75.7432
~53fps

110.7057
112.2314
~49fps

78.62746
80.23053
~38fps

79.31915
80.94699
~50fps

30 Rooms
7 levels

Média de tempo para a geração do mapa: 85.211342 segundos
Média de tempo para a geração dos restantes elementos:86.778038 segundos
Média de fps:~46.6fps

xml leitura

84.73854
85.86366
~52 fps

59.68353
60.92109
~55 fps

72.93908
74.2174
~42 fps

60.59902
61.69869
~47 fps

57.73428
59.10172
~29 fps

30 Rooms
6 levels

Média de tempo para a geração do mapa: 67.13889 segundos
Média de tempo para a geração dos restantes elementos:68.360512 segundos
Média de fps:~45fps

robots

89.18336
90.57423
~56 fps

57.38635
58.66909
~35 fps

92.52403

93.71398
~66 fps

48.14227
49.36386
~53 fps

84.98695
86.15501
~57 fps

30 Rooms
6 levels

Média de tempo para a geração do mapa: 74.444592 segundos
Média de tempo para a geração dos restantes elementos:75.695234 segundos
Média de fps:~53.4fps

Exercise

77.69759
78.96957
~55fps

44.14303
45.36034
~53fps

68.48781
69.70802
~53fps

53.76453
54.77607
~57fps

61.39977
62.58322

~42fps

30 Rooms
6 levels

Média de tempo para a geração do mapa: 61.098546 segundos
Média de tempo para a geração dos restantes elementos:62.279444 segundos
Média de fps:~52fps

characters pieces report

76.22061
77.54665
~40fps

51.52659
52.77646
~52fps

57.72561
58.87104
~53fps

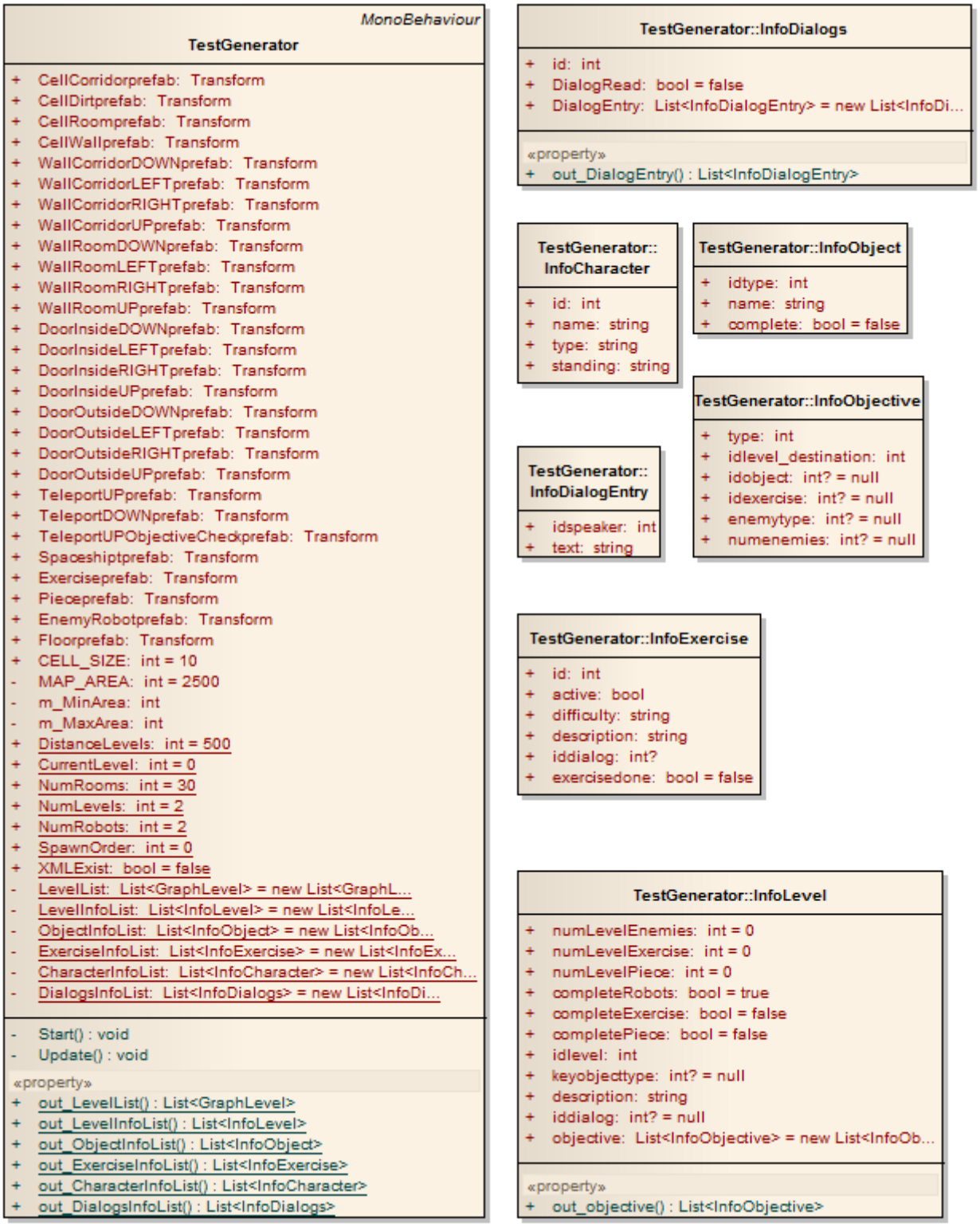
91.22604
92.43787
~45fps

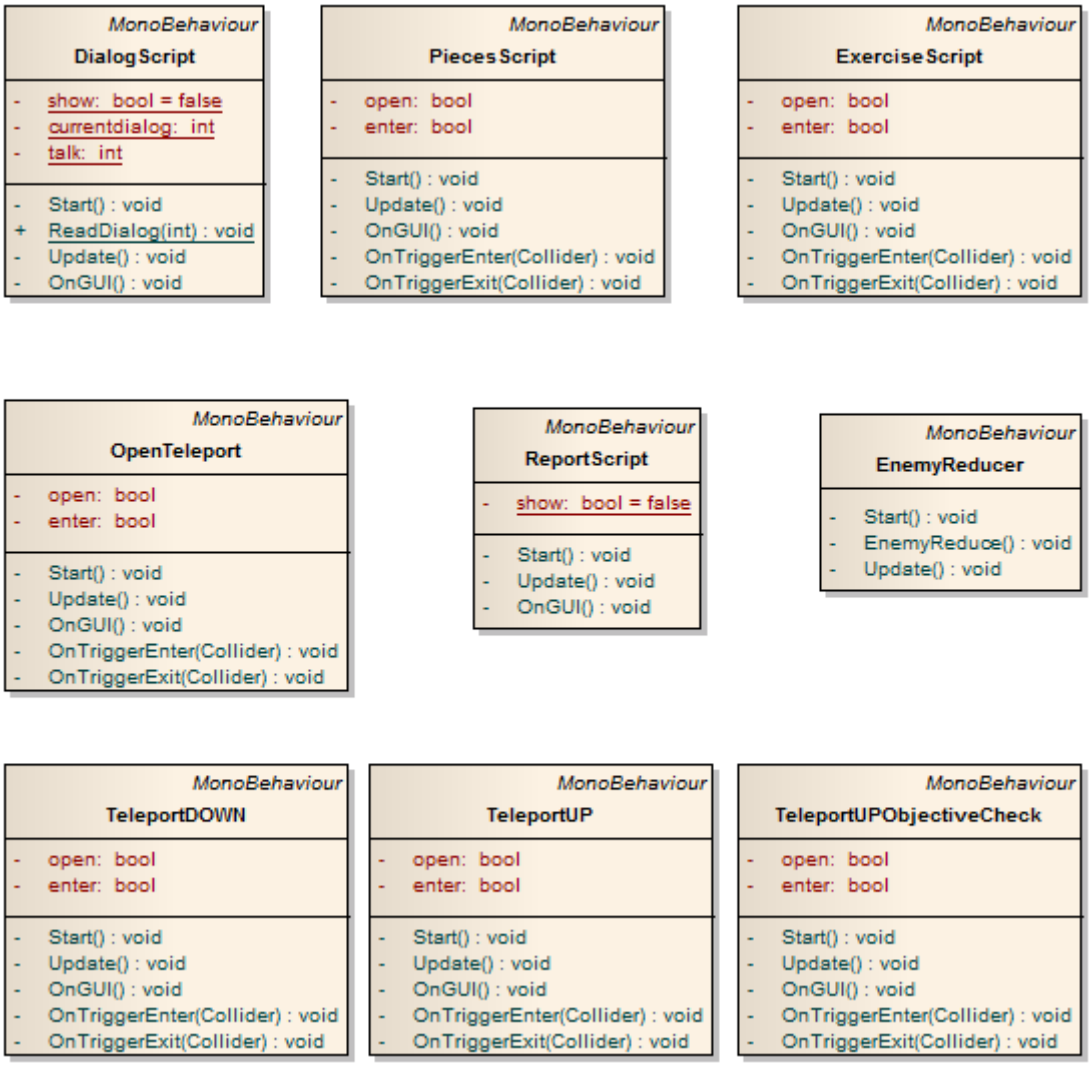
40.61488
42.18255
~32fps

30 Rooms
6 levels

Média de tempo para a geração do mapa: 63.462746 segundos
Média de tempo para a geração dos restantes elementos:64.762914 segundos
Média de fps:~44.4fps

6.3 Organização dos Scripts em C#




MonoBehaviour
DialogScript
- show: bool = false
- currentdialog: int
- talk: int
- Start() : void
+ ReadDialog(int) : void
- Update() : void
- OnGUI() : void
MonoBehaviour
PiecesScript
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void
MonoBehaviour
ExerciseScript
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void
MonoBehaviour
OpenTeleport
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void
MonoBehaviour
ReportScript
- show: bool = false
- Start() : void
- Update() : void
- OnGUI() : void
MonoBehaviour
EnemyReducer
- Start() : void
- EnemyReduce() : void
- Update() : void
MonoBehaviour
TeleportDOWN
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void
MonoBehaviour
TeleportUP
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void
MonoBehaviour
TeleportUPObjectiveCheck
- open: bool
- enter: bool
- Start() : void
- Update() : void
- OnGUI() : void
- OnTriggerEnter(Collider) : void
- OnTriggerExit(Collider) : void

Anexo B

Jogo Orion

Windows 64: https://feupload.fe.up.pt/get/rw9QFTjfQv3Dv4D

Windows 32: https://feupload.fe.up.pt/get/grRy5yI2kcOcK0a